# MANUAL DE DESARROLLO DE DEMO

MAVEN
JAVAFX
JAXB
JDBC (MYSQL y H2)

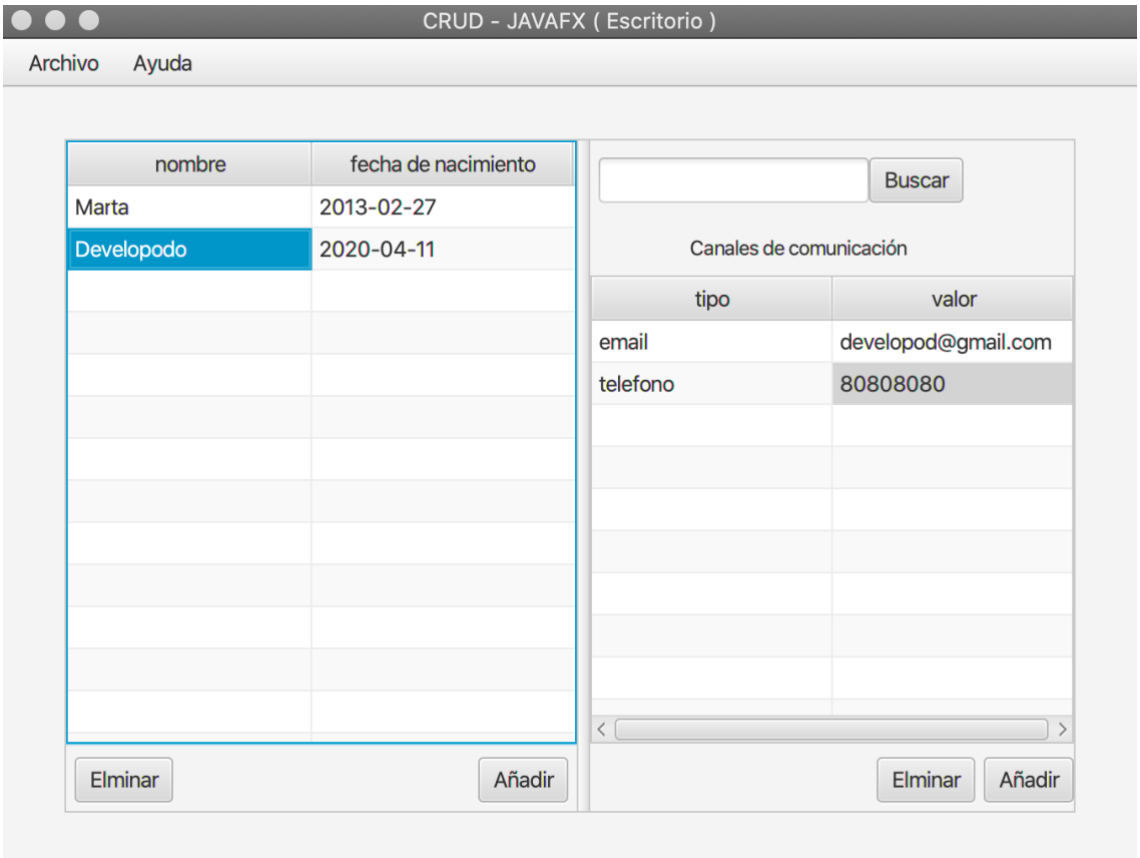
Autor: Carlos Serrano Sánchez
2020

## Tabla de contenido

# 1 INTRODUCCIÓN

Este documento es una manual que guía paso a paso el desarrollo y despliegue de una aplicación standalone para escritorio escrita en Java empleando JavaFX para el diseño de su interfaz.

La aplicación contendrá un menú donde gestionar conexiones a base de datos remotas MySQL o embebidas H2. Estos datos se almacenarán en XML empleando la librería JAXB.
La aplicación principal será un CRUD de contactos a modo de agenda.




Para comenzar a configurar nuestro entorno de desarrollo, debemos dirigirnos a la web official de javaFX: https://openjfx.io/openjfx-docs/#introduction

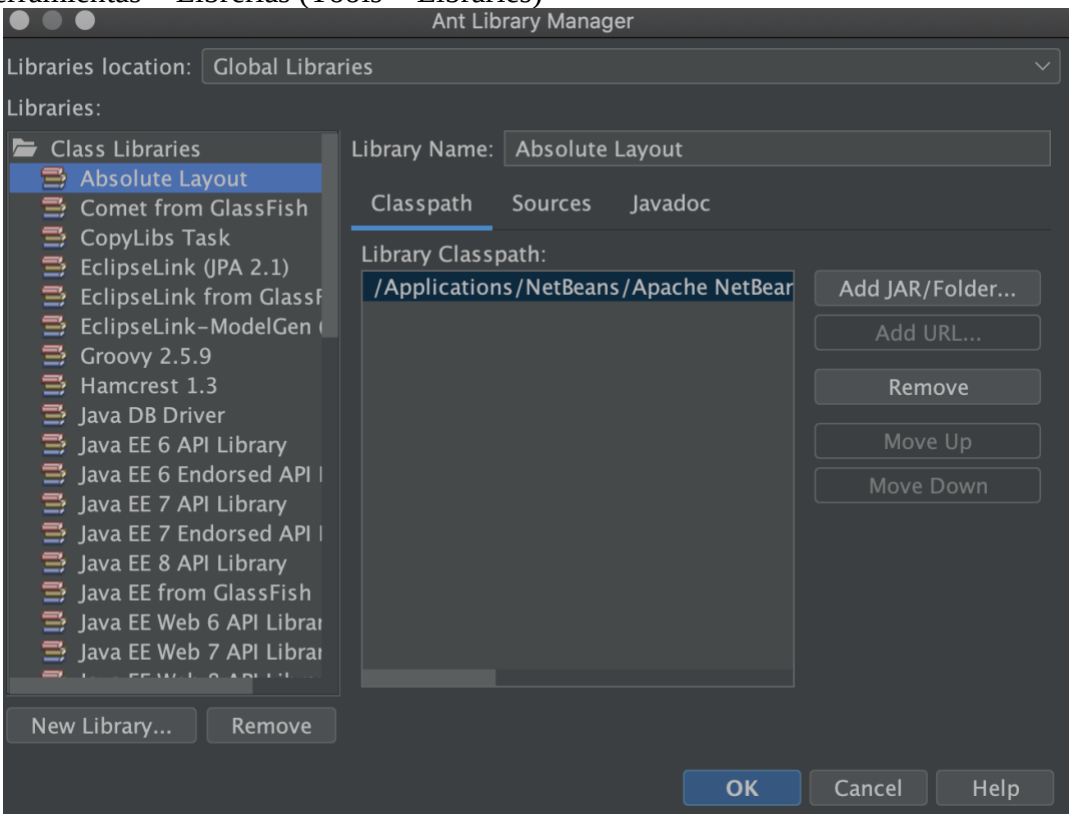
Cosas a comprobar:
Instalar JDK (al menos 13) (http://jdk.java.net/)
Instalar JavaFX JDK (https://gluonhq.com/products/javafx/)
Instalar Scene Builder (https://gluonhq.com/products/scene-builder/)

Vamos a crear dos proyectos, uno básico y otro empleando la herramienta de creación de proyectos Maven.
Todos los códigos: https://github.com/Developodo/JavaFX

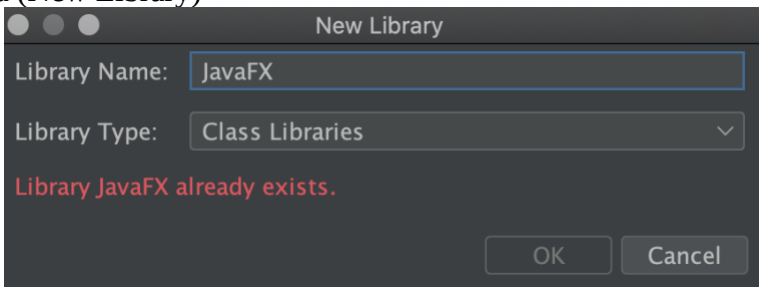
# 2 PARTE 0: JAVAFX

Al descargar el JavaFX JDK habremos copiado (tras descomprinirlo) una carpeta en alguna ruta accesible de nuestra máquina. Ahora debemos incluirla en nuestro IDE.
**Crear una librería nueva JavaFX:**

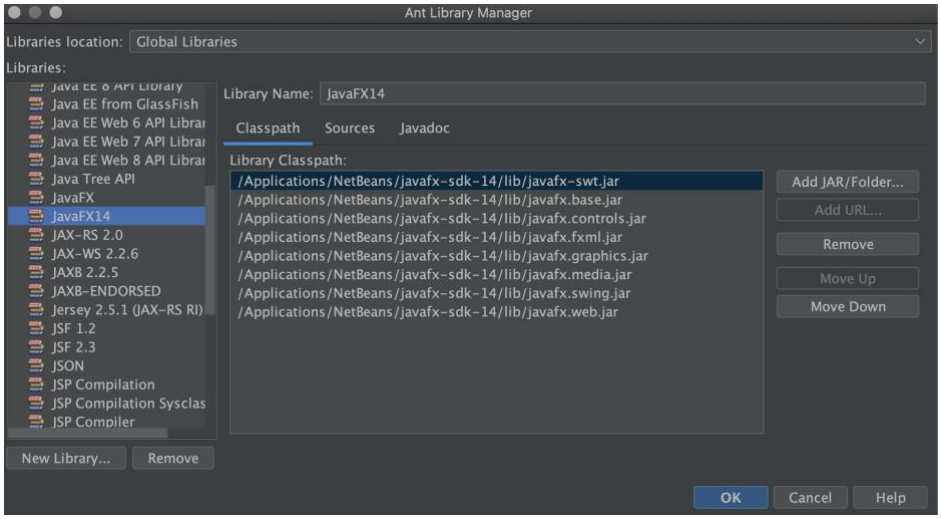
Herramientas > Librerías (Tools > Libraries)



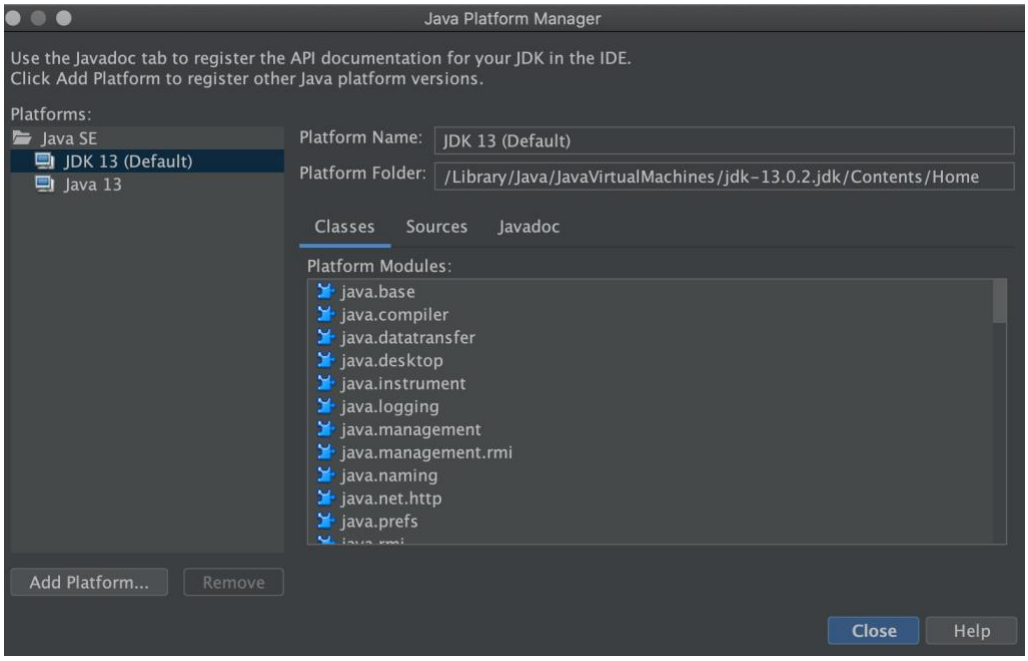

Nueva Librería (New Library)




Asignamos un nuevo y aceptamos. La seleccionamos de la lista y añadimos JAR. Desde ahí buscamos la carpeta donde hemos descargado nuestro JavaFX JDK y añadimos todos los JAR (solo JAR) que se encuentran en su carpeta lib.

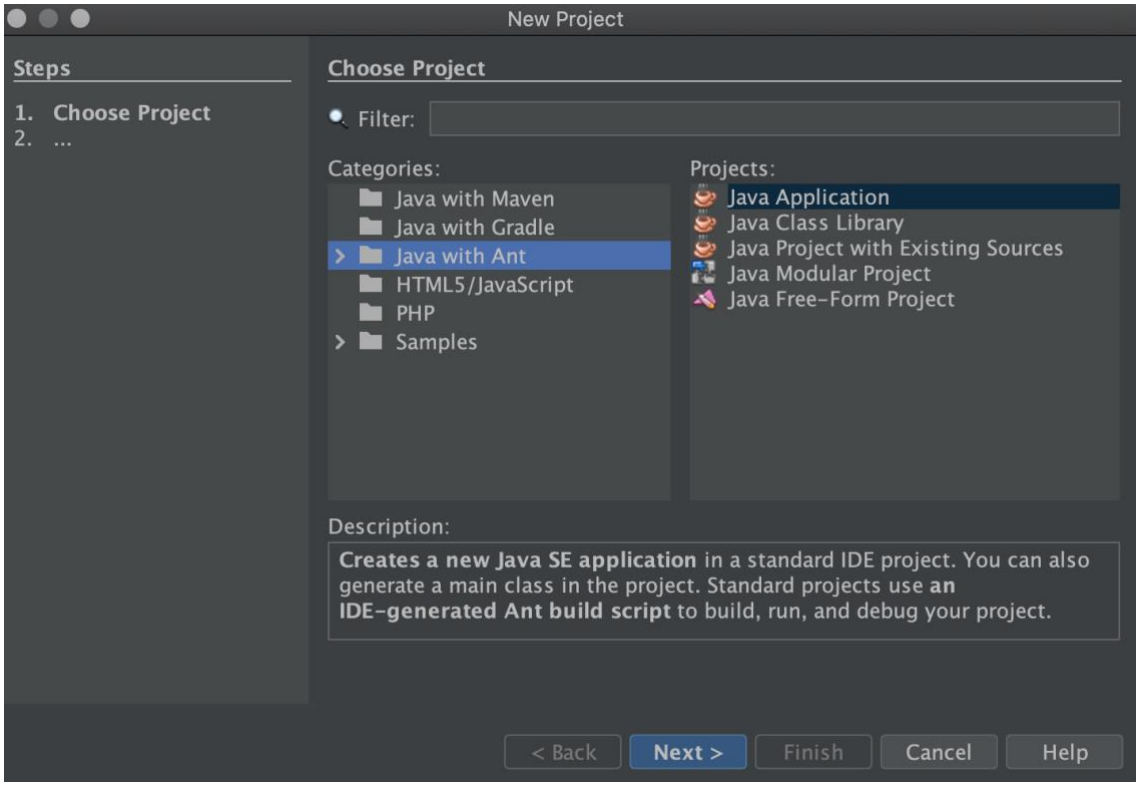



Comprobar que se está empelando la Java JDK adecuada:
Herramientas > Plataformas Java (Tools > Java Platforms)



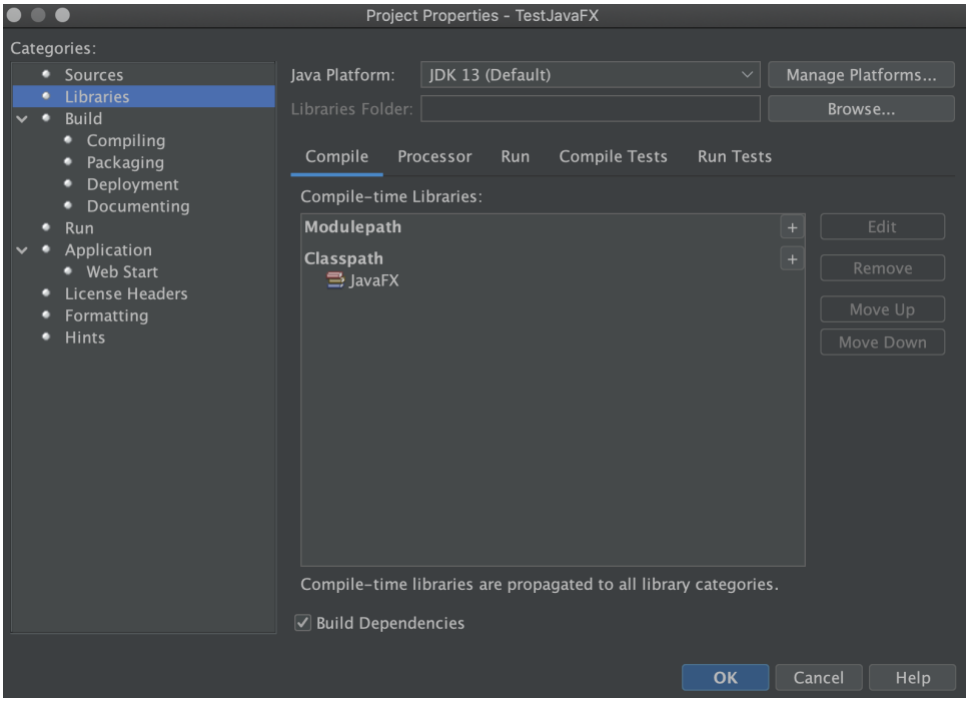

Añadir, si es necesario, la última Java JDK descargada y especificarla como por defecto.
Ya tenemos Netbeans listo.

Crear un proyecto Java Ant en Netbeans

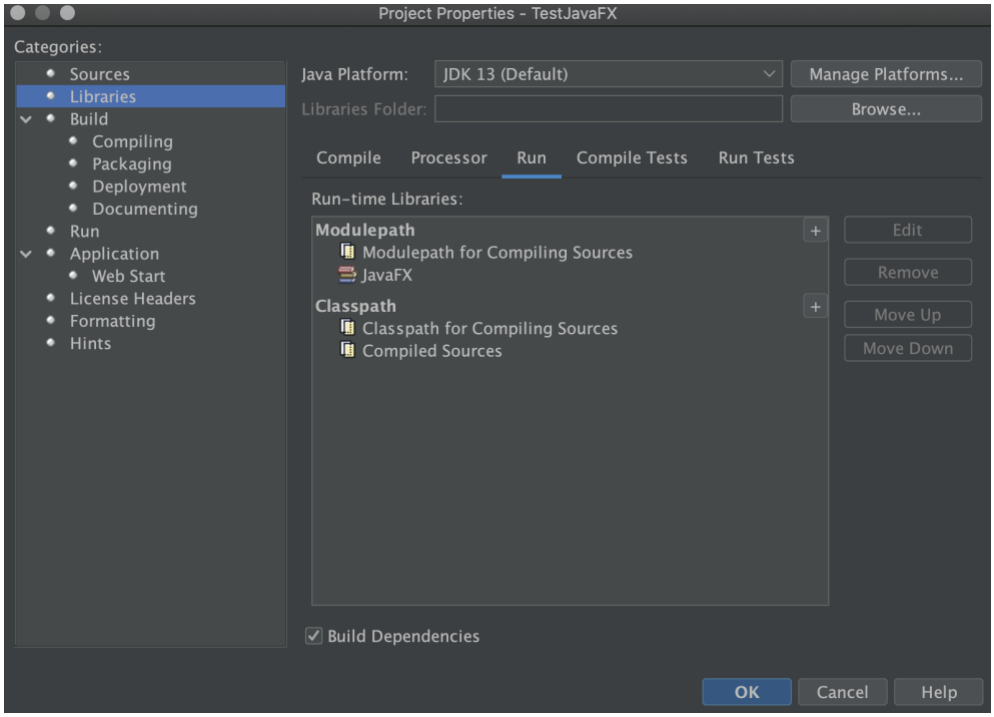



En las propiedades del proyecto establecer los siguientes parámetros:

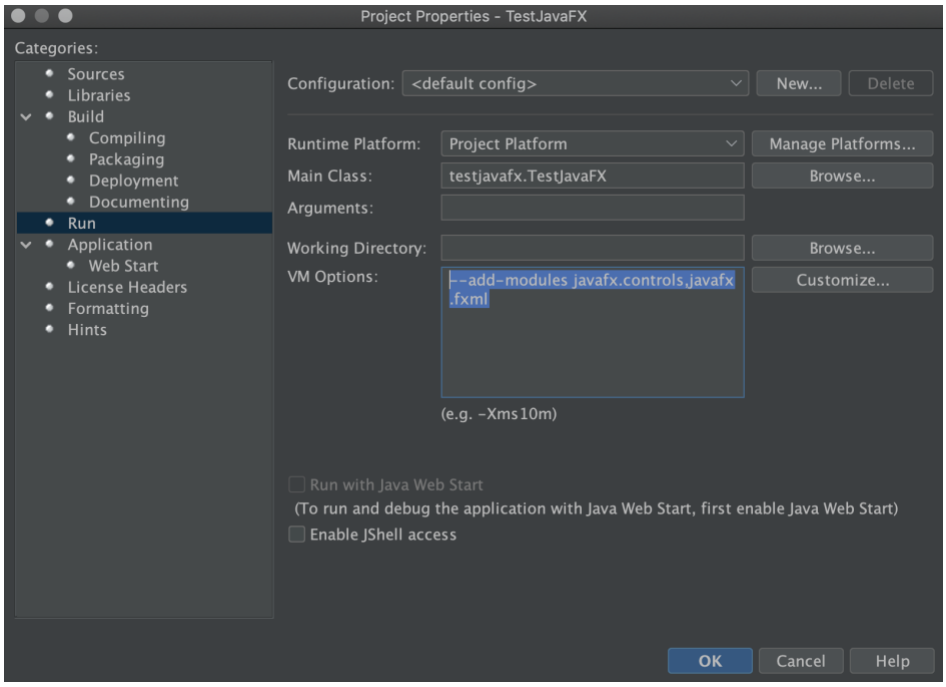
1) En Bibliotecas (Libraries), en la pestaña Compile, añadir en Classpath la librería que hemos creado JavaFX (ojo con el nombre).



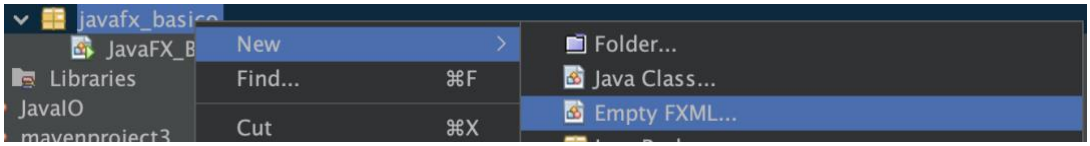
2) En la pestaña Run, añadir en Modulepath la librería JavaFX:



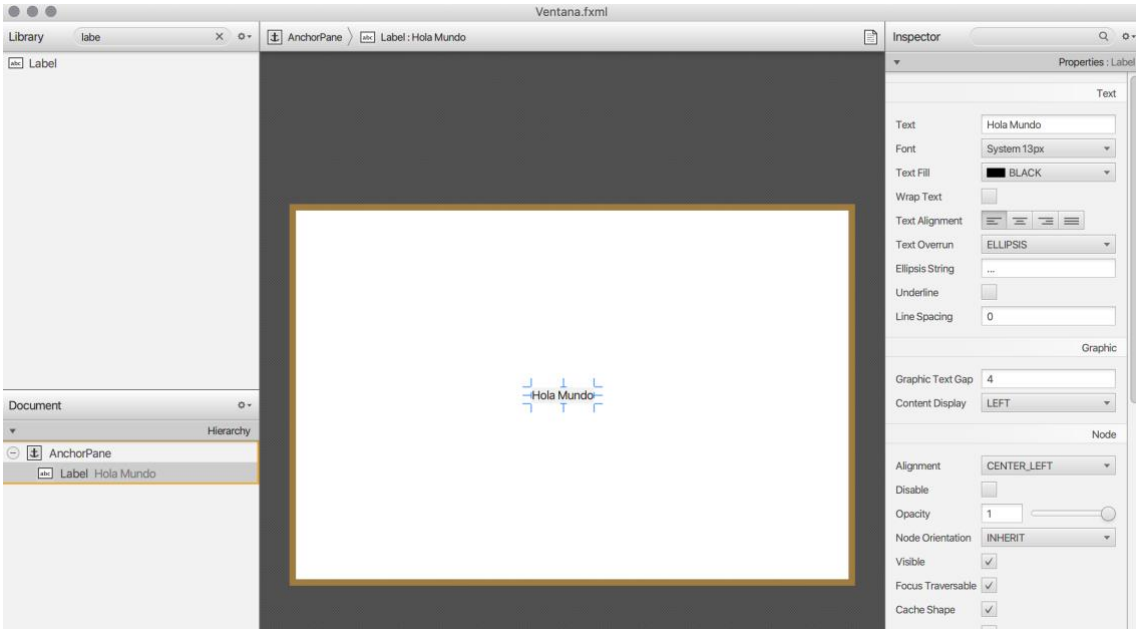

En Run (de las categorías de las propiedades) establecer como VM Options:
--add-modules javafx.controls,javafx.fxml




Creamos un archive vacío de tipo FXML llamado Ventana. En las siguientes opciones elegimos sin controlador y sin hoja CSS.




Hacemos doble clic sobre el archivo y debe abrirse en Scene Builder. Desde esta aplicación mediante Drag&Drop podemos diseñar el espacio. En este caso simplemente crearé un label con la frase Hola Mundo.




Cerramos Scene Builder asegurándonos que hemos guardado los cambios.
En el archivo principal (main) escribimos el siguiente código (Ojo con el nombre del archivo, del paquete y de la clase):

```
package javafx_basico;

import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 *
 * @author carlosserrano
 */
public class JavaFX_Basico extends Application{

  public static void main(String[] args) {
      launch(args);  //call start
    }

    @Override
    public void start(Stage stage) throws Exception {
       Parent root;
```

```
        root = FXMLLoader.load(getClass().getResource("Ventana.fxml"));
        Scene scene= new Scene(root);
        stage.setTitle("HOLA MUNDO");
        stage.setScene(scene);
        stage.show();
    }

}
```

Ahora ejecutamos nuestro proyecto (Play).

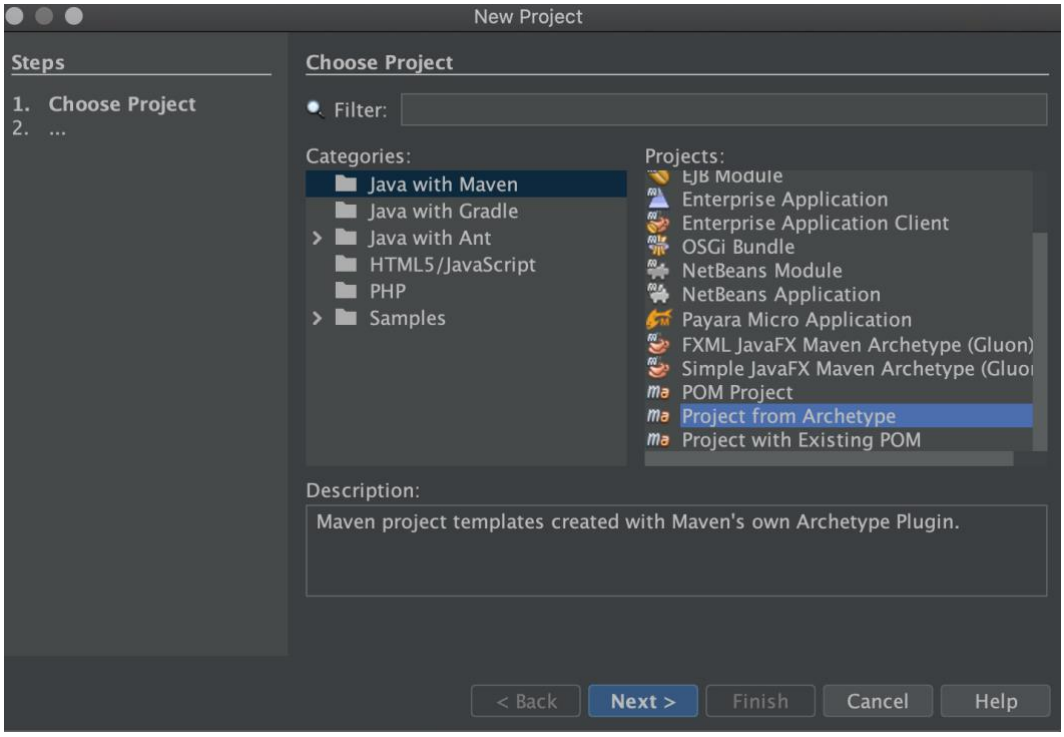

Ya tenemos nuestra aplicación Hola Mundo.

# 3 PARTE 0: JAVAFX Y MAVEN

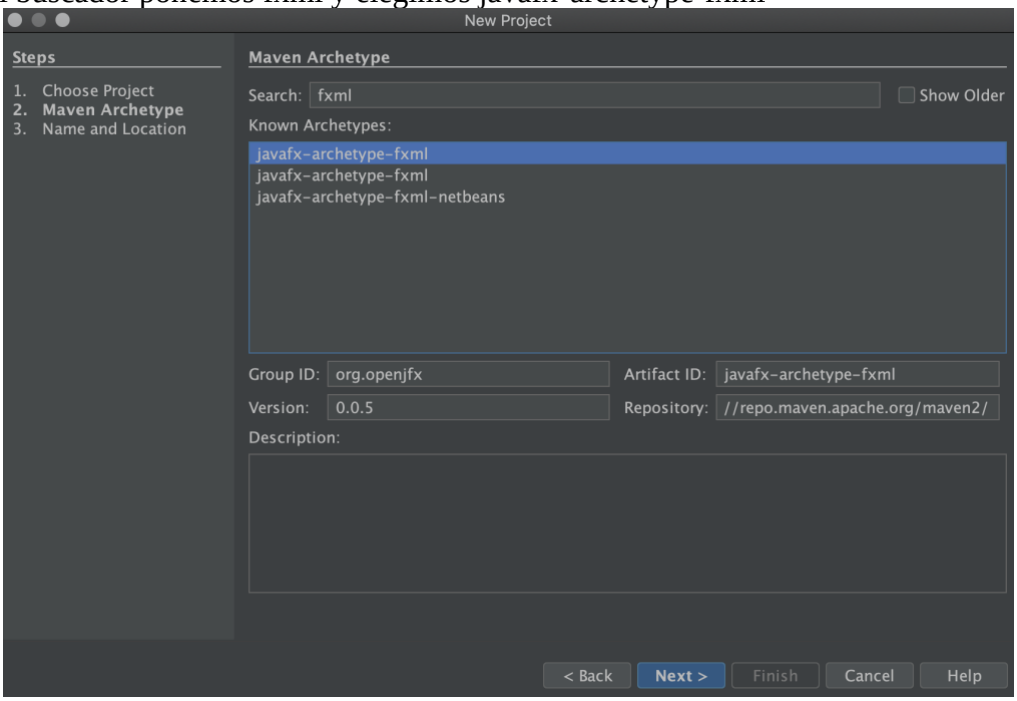
Maven es una herramienta que nos permite gestionar las librerías y dependencias, así como las fases de compilado y testeo en proyectos de cierta envergadura. De hecho, nuestro proyecto empleará esta herramienta, aunque existen otras tan o más populares: Ant, Gradle…

Para saber más de Maven: https://www.genbeta.com/desarrollo/que-es-maven

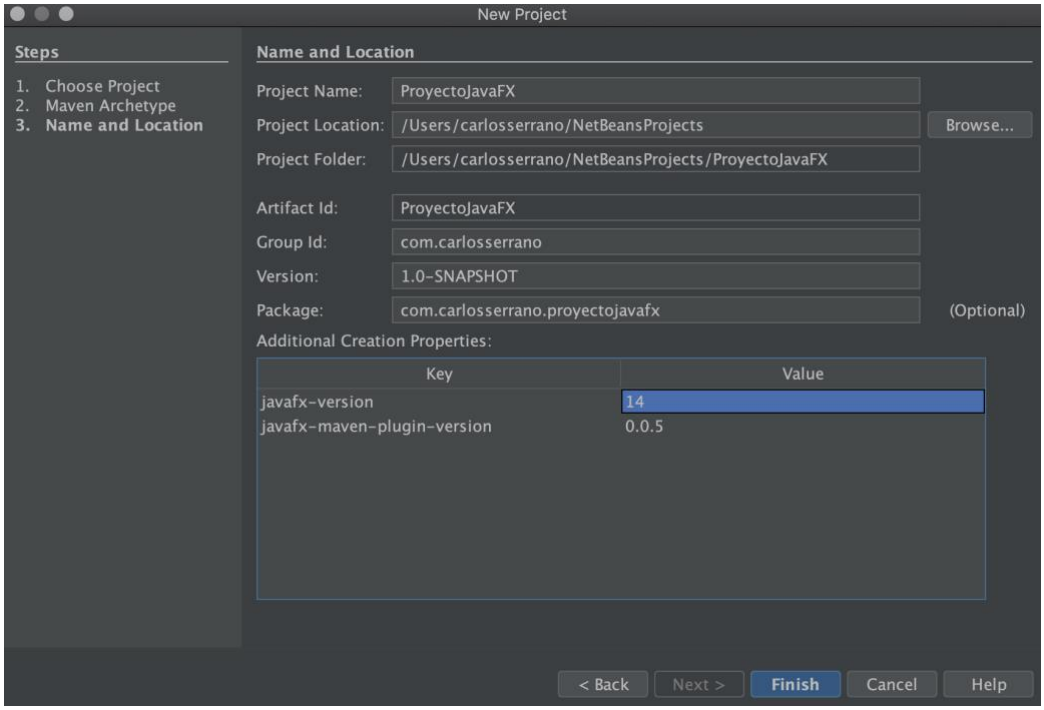
Creamos un proyecto en Netbeans Maven desde Arquetipo:



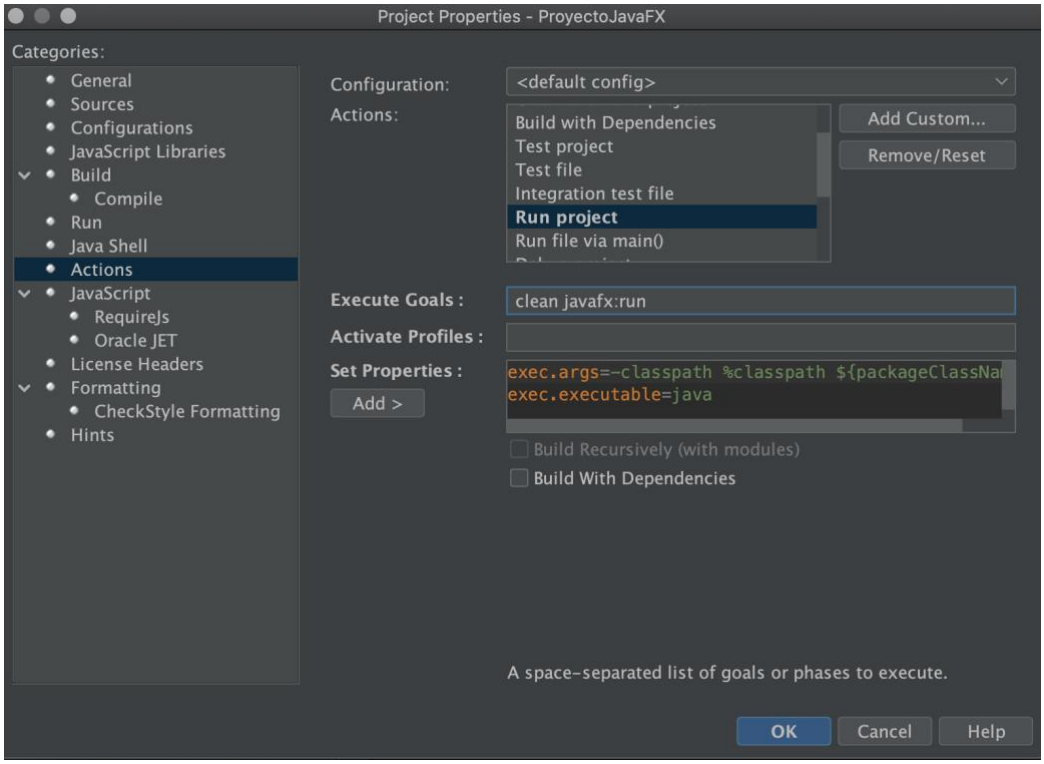

En el buscador ponemos fxml y elegimos javafx-archetype-fxml



En la siguiente ventana elegimos el nombre del proyecto y comprobamos las versiones de javaFX (14):




Una vez creado el proyecto, vamos a sus propiedades y en Actions (Acciones), elegimos Run Project (Ejecutar proyecto) y establecemos como Objetivos (Execute Goals): clean javafx:run

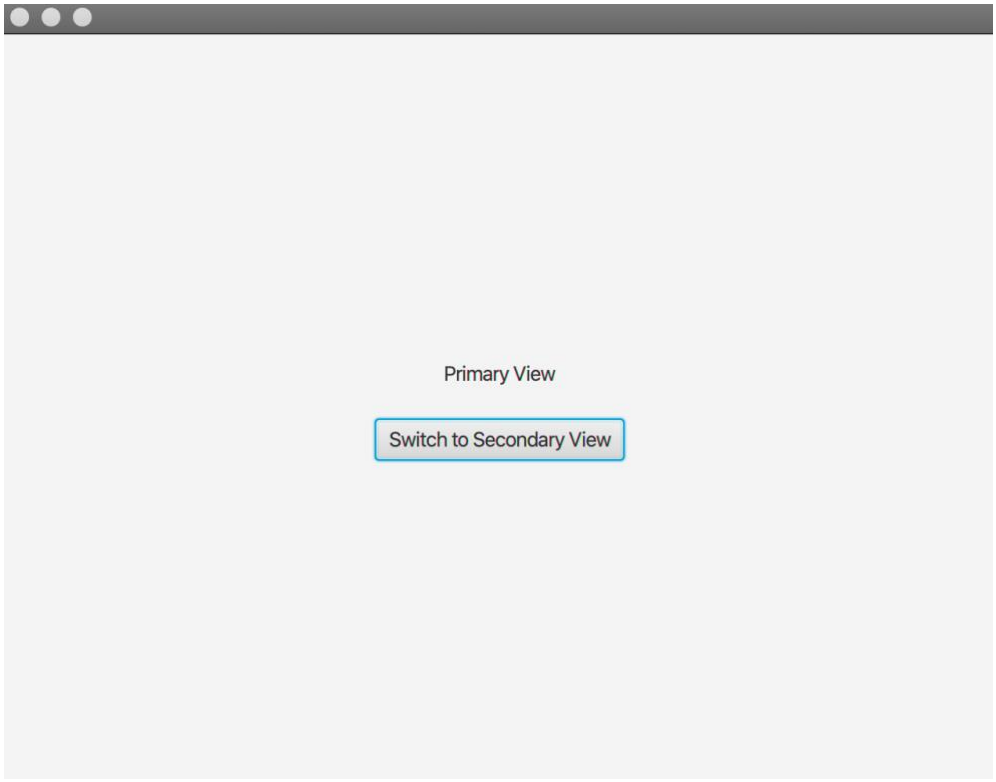


Nos aseguramos de que el archivo Project Files > pom.xml tiene una estructura similar a esta:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.carlosserrano</groupId>
  <artifactId>ProyectoJavaFX</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-controls</artifactId>
      <version>14</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-fxml</artifactId>
      <version>14</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <release>11</release>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.0.4</version>
        <configuration>
          <mainClass>com.carlosserrano.proyectojavafx.App</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Nos aseguramos de que todas las dependencias están descargadas (botón derecho sobre Dependecies y clic sobre Download Declared Dependencies).

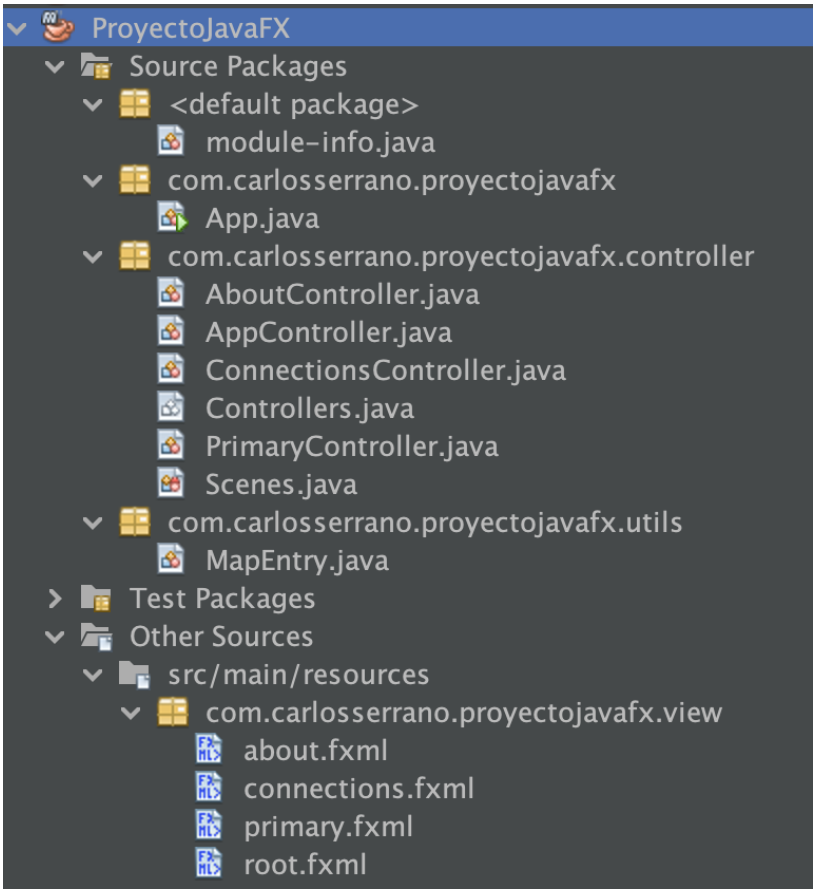
Finalmente, ejecutamos nuestro proyecto (Play):




A partir de esta aplicación construiremos nuestro proyecto.

# 4 PARTE 1: JAVAFX

A partir del proyecto Maven anterior realizamos las modificaciones oportunas para tener la siguiente estructura de carpetas.




A continuación, explicamos las diferentes partes.
Como se observa existen 4 archivos fxml que corresponden con:
Root.fxml será el layout principal de la aplicación, es decir será el que cargaremos y nos servirá de plantilla para inyectar los demás en su parte central. Por tanto, en realidad tendremos 3 páginas: about, connections y primary.

*App.java* es el archivo main, su código es:

```
package com.carlosserrano.proyectojavafx;

import com.carlosserrano.proyectojavafx.controller.AppController;
import com.carlosserrano.proyectojavafx.controller.Controllers;
import com.carlosserrano.proyectojavafx.controller.Scenes;
import com.carlosserrano.proyectojavafx.utils.MapEntry;
import javafx.application.Application;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.io.IOException;
import javafx.scene.layout.BorderPane;
```

```
public class App extends Application {

    public Scene scene;
    public Stage mainStage;
    public BorderPane rootLayout;
    /**
     * Main Controller must be accessible from everywhere
     */
    public AppController controller;

    @Override
    public void start(Stage stage) throws IOException {

        MapEntry<Parent, Controllers> m=AppController.loadFXML(Scenes.ROOT.getUrl());

        mainStage=stage;
        rootLayout=(BorderPane)m.getKey();
        scene = new Scene(rootLayout, 640, 480);
        stage.setScene(scene);

        controller=(AppController)m.getValue();
        controller.setMainApp(this);
        controller.changeScene(Scenes.PRIMARY);
        stage.show();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

En esta ocación, su método start hace uso de un método estático loadFXML de la case AppController que analizaremos a continuación. Se le pasa como parámetro Scener.ROOT.getUrl() que coincidirá como veremos con “view/root”. Una vez cargado el archivo FXML devuelve un MapEntry, clase que veremos con dos objetos en su interior, un parent (que será el layout principal) y con Controllers que será el controlador de ese layout principal.

```
controller=(AppController)m.getValue();
controller.setMainApp(this);
```

Una vez obtiene el controlador con getValue() hace un casting para asignarlo a su variable controller y llama a su método setMain pasándole la referencia de esta clase principal (this).

*AppController.java* es la clase que hará de controlador del layout principal root:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.carlosserrano.proyectojavafx.controller;

import com.carlosserrano.proyectojavafx.App;
import com.carlosserrano.proyectojavafx.utils.MapEntry;
```

```
import java.io.IOException;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.control.MenuItem;

public class AppController extends Controllers{
   //scene actual cargada
   public Scenes backScene;
   public Scenes currentScene;

   /**
    * Recibe la url de una archivo FXML (de la carpeta resources) y devuelve su contenedor y controlador
    * @param fxml url del archivo
    * @return @see(MapEntry) de un contenedor y su controlador si lo tuviera
    * @throws IOException en caso de error de lectura
    */
   public static MapEntry<Parent,Controllers> loadFXML(String fxml) throws IOException {
      FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
      Parent p=fxmlLoader.load();
      Controllers c=fxmlLoader.getController();
      MapEntry<Parent, Controllers> result=new MapEntry<>(p,c);
      return result;
   }

   /**
    * Carga en el Layout de la app principal la escena que se le pase en la
    * zona central. (requisito: el layout principal debe ser borderpane).
    * Le pasa automaticamente al controlador de la escena la referncia a la
    * clase principal para poder tener acceso a su controlador.
    * @param scene La escena a cargar @see(Scenes)
    */
   public void changeScene(Scenes scene){
      try {
         MapEntry<Parent, Controllers> m=loadFXML(scene.getUrl());
         this.app.rootLayout.setCenter(m.getKey());
         if(m.getValue()!=null)
             m.getValue().setMainApp(this.app);
         if(backScene!=currentScene){
            backScene=currentScene;
         }
         this.currentScene=scene;
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
    /**
    * No se usa en este proyecto, dado que nunca se cambiará el rootLayout.
    * Se deja como documentación
    * @param fxml
    * @throws IOException
    */
    private void setRoot(Scenes scene) throws IOException {
      this.backScene=null;
      this.currentScene=null;
      MapEntry<Parent, Controllers> m=loadFXML(scene.getUrl());
      this.app.scene.setRoot(m.getKey());
```

```
        this.app.controller=(AppController)m.getValue();
    }

    @FXML
    private MenuItem con;

    @FXML
    public void connectionsMenu(){
        changeScene(Scenes.CONN);
    }
    @FXML
    public void AboutPage(){
        changeScene(Scenes.ABOUT);
    }
    @FXML
    public void closeApp(){
        System.exit(0);
    }
    @FXML
    public void enableCon(){
        if(con!=null)
        con.setDisable(false);
    }
    @FXML
    public void disableCon(){
        if(con!=null)
        con.setDisable(true);
    }
    @FXML
    public void title(String txt){
        this.app.mainStage.setTitle(txt);
    }

}
```

Como se observa hereda de Controllers que se analiza posteriormente.
Los métodos más relevantes son:

- loadFXML que carga el archivo FXML que se le pase y su controlador, devolviendo un MapEntry.
- changeScene que carga la escena (es un enum con las urls de los diferentes xmls asociándolos a un nombre de escena) y la establece como pantalla centrar del layout principal, se hace acceso a él mediante App. Además mantiene un track de la pantalla anterior por si hay que volver para atrás (solo una vez, se podría mejorar este método con una pila de historial pantallas y permitiría ir hacia atrás más veces).
- Los demás elementos marcados con el modificador @FXML son propiedades o métodos que se llaman desde la interfaz root.fxml como veremos más adelante.

*Controllers.java* es una clase abstracta que heredan todos los controladores:

```
package com.carlosserrano.proyectojavafx.controller;
```

```
import com.carlosserrano.proyectojavafx.App;

public abstract class Controllers {
   App app;
   public void setMainApp(App app){
      this.app=app;
      this.onLoad();
   }
   //To be ovewritten
   void onLoad(){};
}
```

Tiene una propiedad app del tipo de la principal clase de la aplicación y método setMain empleado, como se observó en chageScene para que cuando un FXML es cargado se obtenga su controlador asociado y si existe se le pase una referencia de la aplicación principal ¿Para qué? Para que desde cualquier controlador de la aplicación se pueda acceder a los métodos de la aplicación principal y su controlador, de esta forma está todo comunicado.
Existe además un método onLoad que es llamado en setMain este método se ejecuta siempre que un FXML es cargado y lo podemos emplear como contenedor de tareas que queremos que se hagan en cuanto se presenta una nueva página.

*MapEntry.java* es una clase útil que nos sirve para devolver dos objetos (clave y valor) encapsulador en un Entry. Es decir, solo la empleamos para poder devolver dos pares de objetos en un return, concretamente el método loadFXML de AppController.

```
package com.carlosserrano.proyectojavafx.utils;

import java.util.Map;

public final class MapEntry<K, V> implements Map.Entry<K, V> {
   private final K key;
   private V value;

   public MapEntry(K key, V value) {
      this.key = key;
      this.value = value;
   }

   @Override
   public K getKey() {
      return key;
   }

   @Override
   public V getValue() {
      return value;
   }

   @Override
   public V setValue(V value) {
      V old = this.value;
      this.value = value;
      return old;
```

```
    }

    @Override
    public String toString() {
        return "MapEntry{" + "key=" + key + ", value=" + value + '}';
    }

}
```

Hereda de un Entry convencional de una colección Map para establecer su constructor y sus setter y getter de forma genérica. Tiene un método toString que se ha empleado en la fase de depuración.

Scenes.java es un enum muy peculiar puesto que asocia String a sus entradas. Lo empleamos para ordenar las diferentes pantallas y la url de carga de sus FXML.
package com.carlosserrano.proyectojavafx.controller;

```
public enum Scenes{
    ROOT("view/root"),
    PRIMARY("view/primary"),
    CONN("view/connections"),
    ABOUT("view/about");

    private String url;

    Scenes(String fxmlFile) {
        this.url = fxmlFile;
    }

    public String getUrl() {
        return url;
    }
}
```

Para poder implementarlo se ha creado un constructor peculiar y un getter. Para más información: https://howtodoinjava.com/java/enum/java-enum-string-example/

Nos quedan los controladores de las tres páginas.

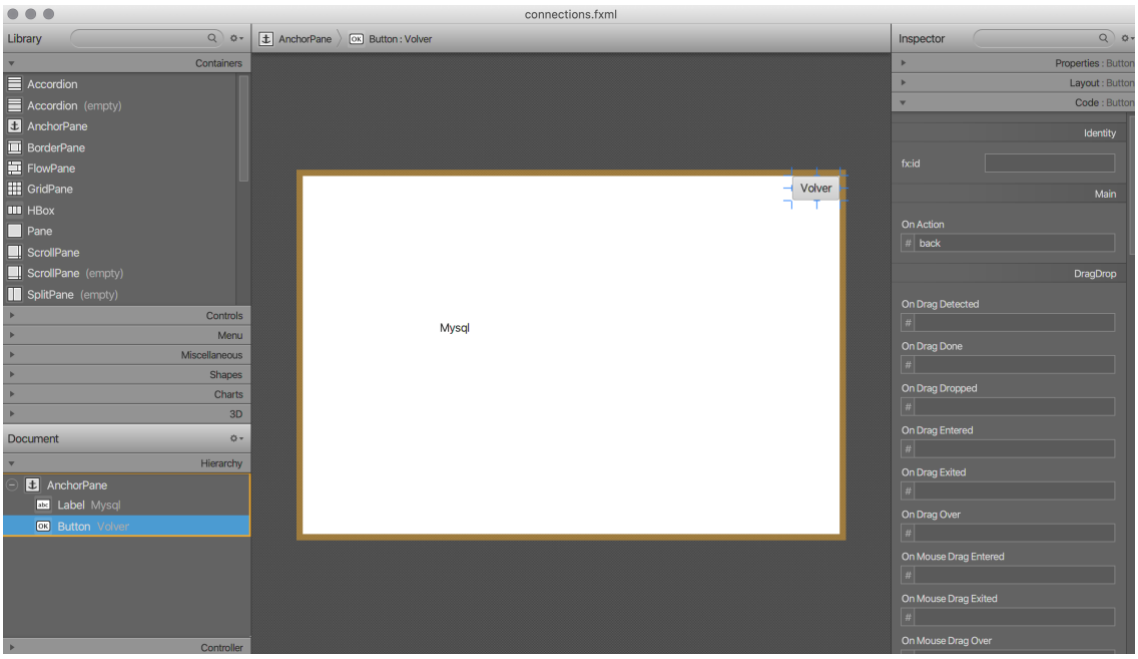
*ConnectionsController.java* corresponde a connection.fxml.

```
package com.carlosserrano.proyectojavafx.controller;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;

public class ConnectionsController extends Controllers implements Initializable {

    @Override
    public void initialize(URL url, ResourceBundle rb) {

    }
```

```
    @FXML
    public void back() {
        this.app.controller.changeScene(Scenes.PRIMARY);
    }

    void onLoad(){
        this.app.controller.disableCon();
        this.app.controller.title("CONEXIONES");
    }
}
```

Implementa Initializable que puede ser útil, aunque no es usada. El método initialize es ejecutado en cuanto es cargada la pantalla, antes incluso de onLoad. Tiene un método back llamado desde la interfaz que permite volver a la pantalla principal. Y onLoad deshabilita un botón del menú y cambia el título de la aplicación.

El archivo *connections.fxml* básicamente tiene un botón al que se le asocia la acción back.




Su código es:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
```

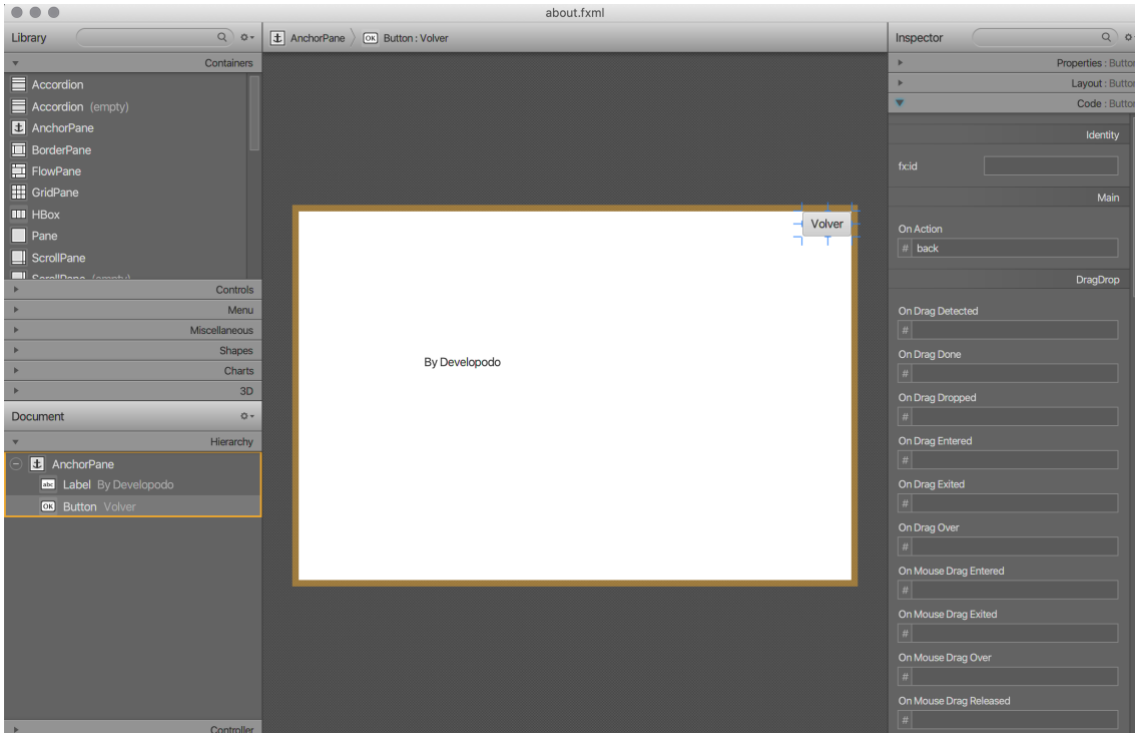
```
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.carlosserrano.proyectojavafx.controller.ConnectionsController">
  <children>
    <Label layoutX="153.0" layoutY="161.0" text="Mysql" />
    <Button layoutX="531.0" layoutY="8.0" mnemonicParsing="false" onAction="#back" text="Volver"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0" />
  </children>
</AnchorPane>
```

El archivo AboutController.java contiene un código similar al anterior:

```
package com.carlosserrano.proyectojavafx.controller;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;

public class AboutController extends Controllers implements Initializable{

   @Override
   public void initialize(URL url, ResourceBundle rb) {
   }

   @FXML
   public void back(){
      if(this.app.controller.backScene!=null){
         this.app.controller.changeScene(this.app.controller.backScene);
      }
   }

}
```

Con la diferencia que la acción back vuelve a la pantalla que estuviera antes cargada.

El archivo about.fxml es prácticamente igual que connections.fxml.




Su código es:

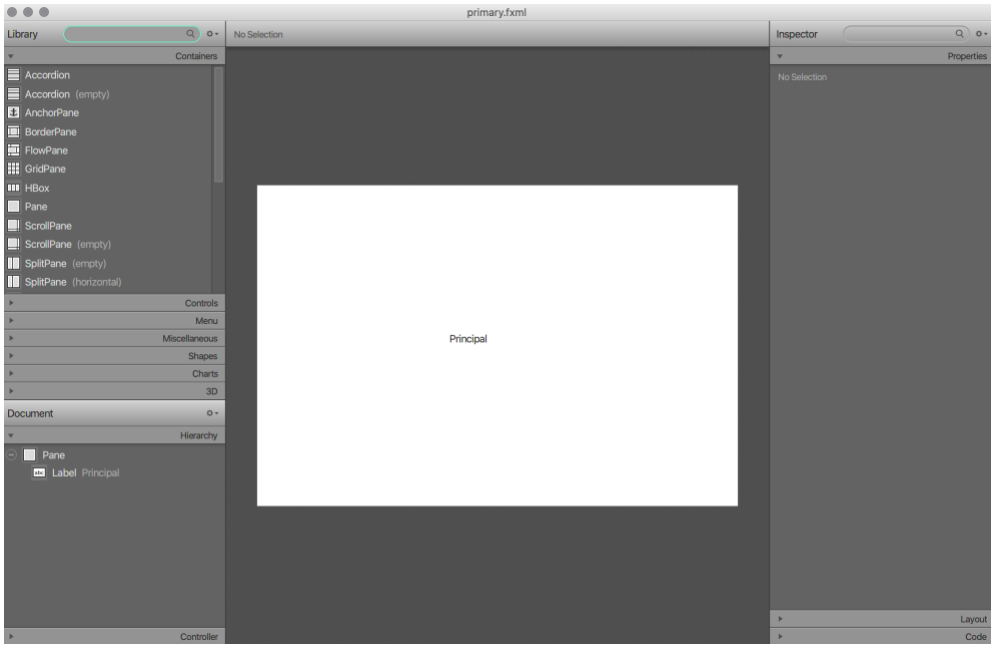
```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.carlosserrano.proyectojavafx.controller.AboutController">
  <children>
    <Label layoutX="136.0" layoutY="155.0" text="By Developodo" />
    <Button layoutX="545.0" layoutY="1.0" mnemonicParsing="false" onAction="#back" text="Volver"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0" />
  </children>
</AnchorPane>
```

El archivo *PrimaryController.java*:

```
package com.carlosserrano.proyectojavafx.controller;

import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.Initializable;

public class PrimaryController extends Controllers implements Initializable{

    @Override
    public void initialize(URL url, ResourceBundle rb) {
```

```
    }

    @Override
    void onLoad() {
        this.app.controller.title("CRUD - JAVAFX");
        this.app.controller.enableCon();
    }

}
```

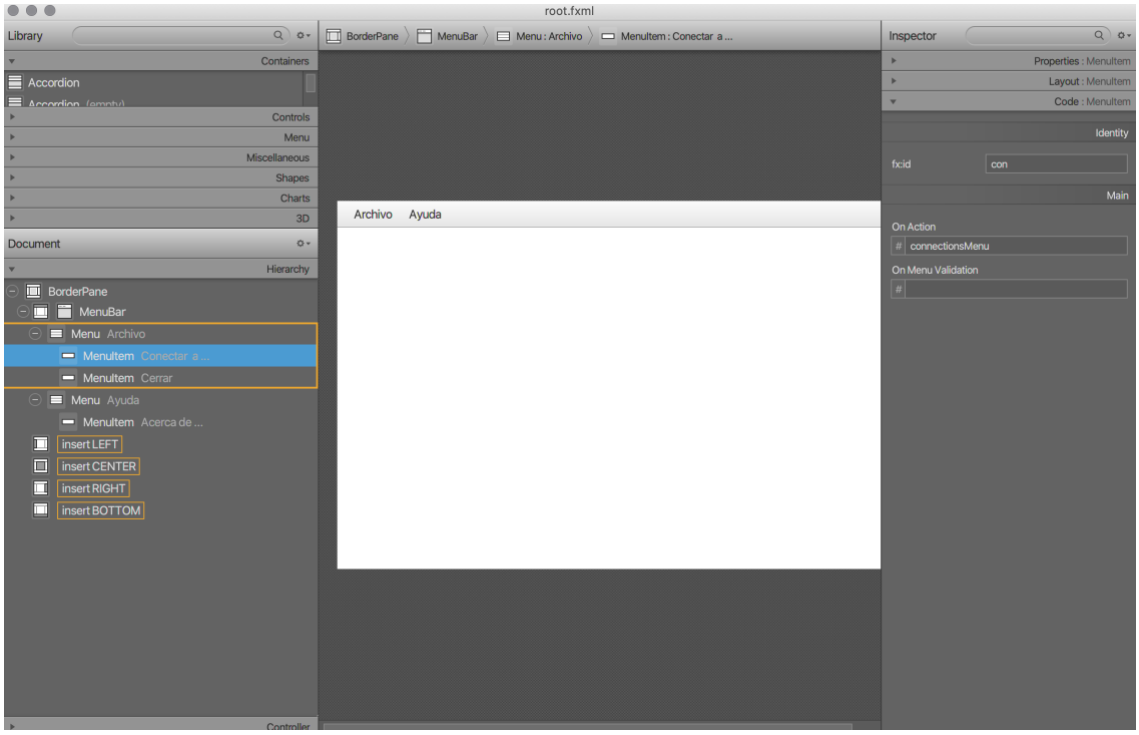
Y su archivo *primary.fxml* está prácticamente vacío:




Su código:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Button?>
<?import javafx.geometry.Insets?>

<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.carlosserrano.proyectojavafx.controller.PrimaryController">
  <children>
    <Label layoutX="240.0" layoutY="183.0" text="Principal" />
  </children>
</Pane>
```

Todos estos FXML se inyectan en *root.fxml*:




Cuya composición sí es más interesante. Su layout principal es un BORDERPANE que permite ubicar elementos en su TOP, LEFT, RIGHT, BOTTOM y CENTER. En su TOP hay una barra de menú (MenuBar) con dos botones de menú, uno de ellos, Archivo, contiene dos Menú Item con acciones asociadas.

Su código es:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.carlosserrano.proyectojavafx.controller.AppController">
  <top>
    <MenuBar BorderPane.alignment="CENTER">
      <menus>
        <Menu mnemonicParsing="false" text="Archivo">
          <items>
            <MenuItem fx:id="con" mnemonicParsing="false" onAction="#connectionsMenu" text="Conectar  a ..." />
            <MenuItem mnemonicParsing="false" onAction="#closeApp" text="Cerrar" />
          </items>
        </Menu>
        <Menu mnemonicParsing="false" text="Ayuda">
          <items>
```

```
            <MenuItem mnemonicParsing="false" onAction="#AboutPage" text="Acerca de ..." />
          </items></Menu>
      </menus>
    </MenuBar>
  </top>
</BorderPane>
```

Solo queda por analizar module-info.java:

```
module com.carlosserrano.proyectojavafx {
    requires javafx.controls;
    requires javafx.fxml;
    requires java.base;

    opens com.carlosserrano.proyectojavafx.controller to javafx.fxml;
    exports com.carlosserrano.proyectojavafx;
}
```

Se ha modificado respecto al original la línea:

```
    opens com.carlosserrano.proyectojavafx.controller to javafx.fxml;
```

Para que desde los archivo FXML (en view) se tenga acceso a los controladores que están en com.carlosserrano.proyectojavafx.controller.

El código de esta versión se encuentra en el respositorio: https://github.com/Developodo/JavaFX_( (ProyectoJavaFX_P1)

# 5 PARTE 2: JAVAFX y JAXB

En esta parte vamos a desarrollar la pantalla de conexiones. Tendrá un menú donde podremos crear, editar o borrar conexiones (a servidores de bases de datos). Este CRUD almacenará su información en un archivo XML. Además, nos permitirá seleccionar la conexión que deseamos emplear en cada momento y se almacenará en las preferencias del usuario para que sea recordada la próxima vez que se ejecute la aplicación.

Vamos a usar la librería JAXB que nos permite pasar de objetos a su representación XML (Marshall) y viceversa (UnMarshall)

Copiamos su depencia para Maven:

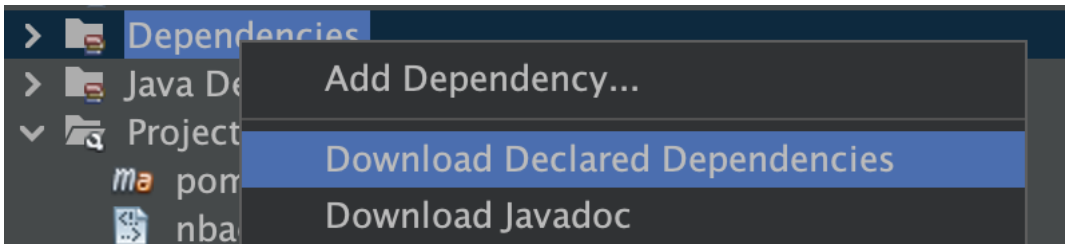
```
<dependency>
        <groupId>jakarta.xml.bind</groupId>
        <artifactId>jakarta.xml.bind-api</artifactId>
        <version>2.3.3</version>
    </dependency>
```

Lo copiamos en la sección dependecies de pom.xml, quedando así:

```
  <dependencies>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-controls</artifactId>
      <version>14</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-fxml</artifactId>
      <version>14</version>
    </dependency>
<dependency>
 <groupId>jakarta.xml.bind</groupId>
 <artifactId>jakarta.xml.bind-api</artifactId>
 <version>2.3.3</version>
</dependency>
</dependencies>
```

Esperamos a que se descarguen las dependencias, en cualquier caso, se puede comprobar limpiando y realizando un build de la app:

O las forzamos con botón derecho sobre la carpeta Dependencies:



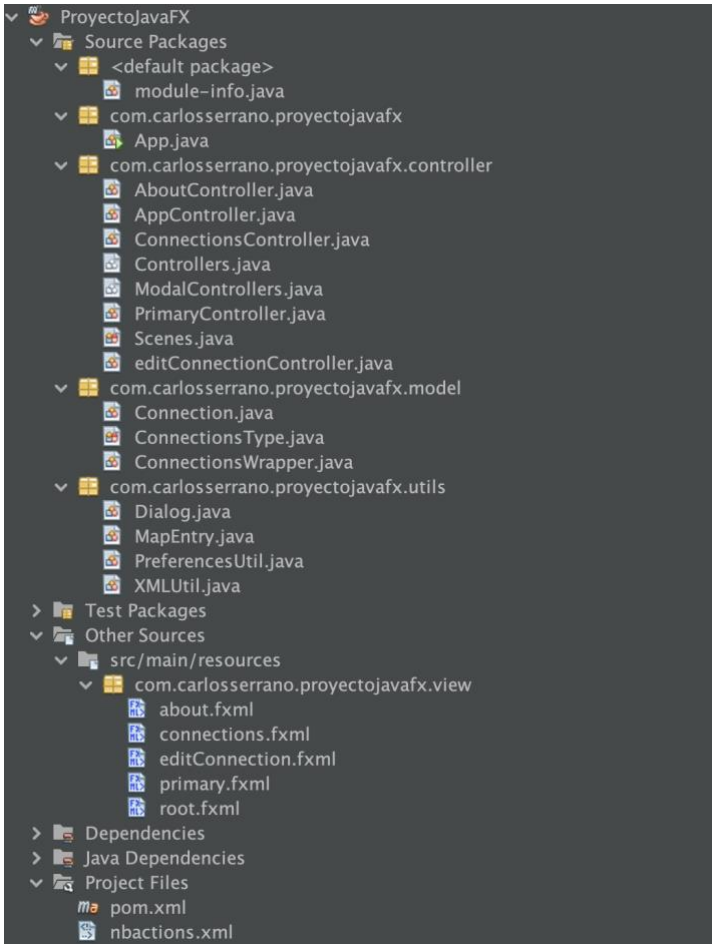

Ahora en el archivo module-info.java incorporamos la nueva librería (es necesario por motivos del arquetipo javafx que hemos creado):

```
module com.carlosserrano.proyectojavafx {
  requires javafx.controls;
  requires javafx.fxml;
  requires java.base;
  requires java.xml.bind;
  requires java.prefs;

  opens com.carlosserrano.proyectojavafx.utils to java.xml.bind; //Para que JAXB pueda ejecutarse en XMLUtil
opens com.carlosserrano.proyectojavafx.model to java.xml.bind; //Para que JAXB pueda ejecutarse en ConnectionWrapper
  opens com.carlosserrano.proyectojavafx.controller to javafx.fxml;

  exports com.carlosserrano.proyectojavafx;
```

```
    exports com.carlosserrano.proyectojavafx.model;  //para que JAXB pueda acceder a Connection y ConnectionWrapper
}
```

A partir de aquí tenemos acceso a la nueva librería.

La estructura de archivos de la versión final de esta parte es:




El archivo *App.java* no ha sido modificado ni *root.fxml* tampoco.

Comenzamos por el paquete *com.carlosserrano.proyectojavafx.model*. En esta carpeta guardamos todas las clases que están relacionadas con datos a almacenar o tratar.

*Connection.java*

```
package com.carlosserrano.proyectojavafx.model;

import java.util.Objects;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;


public class Connection{
   public static ConnectionsType connectionTypes;

   private StringProperty name;
   private String type;
   private String server;
   private String userName;
   private String password;

   public Connection(String name){
      this.name=new SimpleStringProperty(name);
```

```
        this.type=ConnectionsType.MYSQL.getType();
        this.server="";
        this.userName="";
        this.password="";
    }

    public Connection(){
        this("");
    }
    public StringProperty getN(){
        return name;
    }

    public String getName() {
        return name.getValue();
    }

    public void setName(String name) {
        this.name = new SimpleStringProperty(name);
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getServer() {
        return server;
    }

    public void setServer(String server) {
        this.server = server;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "Connection{" + "name=" + name + ", type=" + type + ", server=" + server + ", userName=" +
userName + ", password=" + password + '}';
```

```
    }

    @Override
    public boolean equals(Object o){
        if(o==this){
            return true;
        }else{
            if(o instanceof Connection){
                Connection other=(Connection)o;
                if(name.getValue().equals(((Connection) o).name.getValue())){
                    return true;
                }else{
                    return false;
                }
            }else{
                return false;
            }
        }
    }

    @Override
    public int hashCode() {
        int hash = 7;
        hash = 97 * hash + Objects.hashCode(this.name);
        return hash;
    }
}
```

Es una clase básica que nos sirve para caracterizar a las conexiones por su nombre, servidor, tipo, usuario y contraseña. Hemos establecido que dos conexiones son iguales si el nombre asignado es el mismo.
El nombre se puede observar que es de tipo StringProperty. Es una clase que nos permite realizar binding con la interfaz gráfica, es decir los cambios se reflejan directamente en los elementos visuales. Para poder realizar la conversión adecuadamente se han creado dos getter de esta variable, uno que devuelve este tipo tan especial y otro, el convencional, que realiza su conversión a String y lo empleamos, no solo para comparar, sino para convertir a XML.

El tipo de conexión es un enum customizado que se muestra a continuación.

*ConnectionsType.java:*

```
package com.carlosserrano.proyectojavafx.model;

public enum ConnectionsType {

    MYSQL("mySQL"),
    H2("H2");

    private String type;

    private ConnectionsType(String type) {
        this.type=type;
    }
```

```
    public String getType(){
        return this.type;
    }
}
```

Tendremos una lista de conexiones, dado que deseamos que el usuario pueda añadir tantas como quiera. Esta lista la almacenaremos en un archivo XML mediante un proceso semiautomático mediante la librería JAXB. No obstante, como veremos más adelante la lista será algo especial y no de las Collections habituales. Por tanto, vamos a crear un Wrapper que nos permita listar estas conexiones y almacenarlas de una vez en un XML.

*ConnectionsWrapper.java:*

```
package com.carlosserrano.proyectojavafx.model;

import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="conns")
@XmlAccessorType(XmlAccessType.FIELD)
public class ConnectionsWrapper {

    @XmlElement(name="conn")
    private List<Connection> conns;

    public List<Connection> getConns() {
        return conns;
    }

    public void setConns(List<Connection> conns) {
        this.conns = conns;
    }

    @Override
    public String toString() {
        return "ConnectionsWrapper{" + "conns=" + conns + '}';
    }
}
```

Como se puede observar no es más que una clase que contiene una lista de Connections. Pero hay algo muy importante:

- @XmlRootElement(name=”conns”) indica que el nodo raíz del XML será conns.
- @XmlAccessorType(XmlAccessType.FIELD) indica que creará un nodo por cada propiedad de las clases, si no se indica lo contrario.
- @XmlElement(name="conn") indica que al iterar la lista creará un nodo conn por cada elemento.

El XML resultante tendrá un aspecto similar a este:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<conns>
  <conn>
    <name>Mi servidor</name>
    <password></password>
    <server>localhost</server>
    <type>mySQL</type>
    <userName>root</userName>
  </conn>
  <conn>
    <name>Embebido</name>
    <password></password>
    <server>~/</server>
    <type>H2</type>
    <userName>sa</userName>
  </conn>
</conns>
```

¿Cómo se realiza toda esta conversión? Sigamos con el paquete *com.carlosserrano.proyectojavafx.utils*

*XMLUtil.java*

```
package com.carlosserrano.proyectojavafx.utils;

import com.carlosserrano.proyectojavafx.model.ConnectionsWrapper;
import com.carlosserrano.proyectojavafx.model.Connection;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;


public class XMLUtil {
   public static String file="conn.xml";

   public static List<Connection> loadDataXML() {
      List<Connection> result=new ArrayList<>();
      File f=new File(file);
      if(f.canRead()){
         try{
            JAXBContext context=JAXBContext.newInstance(ConnectionsWrapper.class);
            Unmarshaller um = context.createUnmarshaller();
            ConnectionsWrapper wrapper = (ConnectionsWrapper) um.unmarshal(f);
            result.addAll(wrapper.getConns());
         }catch(JAXBException ex){
            ex.printStackTrace();
            Dialog.showError("ERROR", "Error writing "+file, ex.toString());
            result=new ArrayList<>();
         }
      }
      return result;
   }
```

```
    public static void writeDataXML(List<Connection> data){
        JAXBContext context;
        try {
            context = JAXBContext.newInstance(ConnectionsWrapper.class);
            Marshaller m=context.createMarshaller();
            m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
            ConnectionsWrapper wrapper = new ConnectionsWrapper();
            wrapper.setConns(data);
            m.marshal(wrapper, new File(file));
        } catch (JAXBException ex) {
            ex.printStackTrace();
            Dialog.showError("ERROR", "Error reading "+file, ex.toString());
        }

    }
}
```

Esta clase tiene dos métodos estáticos que nos permiten hacer el proceso Marshall (objeto > XML) y UnMarshall (XML > objeto) en el archivo conn.xml.
En caso de error muestra un menú de diálogo indicando el error, lo realiza mediante la siguiente clase:

*Dialog.java*

```
package com.carlosserrano.proyectojavafx.utils;

import javafx.scene.control.Alert;


public class Dialog {
    public static void showError(String title, String header,String description){
        showDialog(Alert.AlertType.ERROR, title, header, description);
    }
    public static void showWarning(String title, String header,String description){
        showDialog(Alert.AlertType.WARNING, title, header, description);
    }
    public static void showConfirm(String title, String header,String description){
        showDialog(Alert.AlertType.CONFIRMATION, title, header, description);
    }
    public static void showDialog(Alert.AlertType type, String title, String header,String description){
        Alert alert =new Alert(type);
        alert.setTitle(title);
        alert.setHeaderText(header);
        alert.setContentText(description);
        alert.showAndWait();

    }
}
```

Proporciona tres métodos: para avisos, errores y notificaciones. Podría mejorarse incluyendo confirmación.

*MapEntry.java* ya se comentó en la parte anterior y no ha sido modificado.

*PreferencesUtil.java* lo empleamos para almacenar propiedades (clave y valor en String) en local, en la máquina del usuario. De esta forma podemos guardar preferencias a modo de cookie que se mantendrán de una ejecución a otra de la aplicación.

```
import com.carlosserrano.proyectojavafx.App;
import com.carlosserrano.proyectojavafx.model.Connection;
import java.util.List;
import java.util.prefs.Preferences;

public class PreferencesUtil {
   public static Connection getPreference(){
      Connection result=null;
      Preferences prefs= Preferences.userNodeForPackage(App.class);
      String nameC=prefs.get("conn", null);
      if(nameC!=null){
         List<Connection> conns=XMLUtil.loadDataXML();
         Connection search=new Connection(nameC);
         int index=conns.indexOf(search);
         if(index>-1){
            result=conns.get(index);
         }else{
            setPreference(null);
         }
      }

      return result;
   }

   public static void setPreference(String nameC){
      Preferences prefs= Preferences.userNodeForPackage(App.class);
      if(nameC!=null){
         prefs.put("conn",nameC);
      }else{
         prefs.remove("conn");
      }

   }
}
```

En este caso guarda en la propiedad conn el nombre de la conexión que elijamos como predefinida. Se emplea App.class (main) como archivo de referencia para guardar dicha propiedad.

Fijémonos ahora en el paquete *com.carlosserrano.proyectojavafx.controller*, sin duda el más interesante y complejo hasta el momento.

*Controllers.java,* es la clase que heredan todos los controladores para compartir ciertas propiedades, como es la referencia a la clase principal.

```
package com.carlosserrano.proyectojavafx.controller;

import com.carlosserrano.proyectojavafx.App;

public abstract class Controllers {
   App app;
```

```
    public void setMainApp(App app){
        this.app=app;
        this.onLoad();
    }
    //To be ovewritten
    void onLoad(){};
    //To be ovewritten
    void doOnCloseModal(Object response){}
}
```

Se ha añadido un método doOnCloseModal que será ejecutado automáticamente cuando una ventana Modal (superpuesta) se cierra. Lo emplearemos para enviar información de vuelta (Modal > App).

*ModalControllers.java* es la clase de controlador que tendrán los modales. Hereda de Controllers, pero añade ciertas características.

```
package com.carlosserrano.proyectojavafx.controller;

import javafx.stage.Stage;

public abstract class ModalControllers extends Controllers{
    Controllers parentController;
    Stage stage;

    public Controllers getParentController() {
        return parentController;
    }

    public void setParentController(Controllers parentController) {
        this.parentController = parentController;
    }

    public Stage getStage() {
        return stage;
    }

    public void setStage(Stage stage) {
        this.stage = stage;
    }
    abstract public void setParams(Object p);

}
```

Tiene una referencia al controlador de la ventana que lo abrió y a su propia stace para poder autocerrarse.
Además, el método setParams, permite enviar info de la App al Modal. Es decir:

- App > Modal , mediante setParams
- Modal > APP, mediante doOnCloseModal

*Scenes.java*, hemos añadido una nueva pantalla:

```
package com.carlosserrano.proyectojavafx.controller;
/**
```

```
 * Enum with String values: https://howtodoinjava.com/java/enum/java-enum-string-example/
 */
public enum Scenes{
    ROOT("view/root"),
    PRIMARY("view/primary"),
    CONN("view/connections"),
    ABOUT("view/about"),
    EDIT("view/editConnection");

    private String url;

    Scenes(String fxmlFile) {
        this.url = fxmlFile;
    }

    public String getUrl() {
        return url;
    }
}
```

*AboutController.java* no ha sido modificado. *about.fxml* tampoco.

*AppController.java*

```
package com.carlosserrano.proyectojavafx.controller;

import com.carlosserrano.proyectojavafx.App;
import com.carlosserrano.proyectojavafx.model.Connection;
import com.carlosserrano.proyectojavafx.utils.MapEntry;
import com.carlosserrano.proyectojavafx.utils.PreferencesUtil;
import java.io.IOException;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.MenuItem;
import javafx.stage.Modality;
import javafx.stage.Stage;

public class AppController extends Controllers{
    //scene actual cargada
    public Scenes backScene;
    public Scenes currentScene;

    public Connection currentConnection;

    @Override
    void onLoad() {
        currentConnection=PreferencesUtil.getPreference();
    }

    /**
     * Recibe la url de una archivo FXML (de la carpeta resources) y devuelve su contenedor y controlador
     * @param fxml url del archivo
     * @return @see(MapEntry) de un contenedor y su controlador si lo tuviera
     * @throws IOException en caso de error de lectura
```

```
 */
public static MapEntry<Parent,Controllers> loadFXML(String fxml) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
    Parent p=fxmlLoader.load();
    Controllers c=fxmlLoader.getController();
    MapEntry<Parent, Controllers> result=new MapEntry<>(p,c);
    return result;
}

/**
 * Carga en el Layout de la app principal la escena que se le pase en la
 * zona central. (requisito: el layout principal debe ser borderpane).
 * Le pasa automaticamente al controlador de la escena la referncia a la
 * clase principal para poder tener acceso a su controlador.
 * @param scene La escena a cargar @see(Scenes)
 */
public void changeScene(Scenes scene){
    try {
        MapEntry<Parent, Controllers> m=loadFXML(scene.getUrl());
        this.app.rootLayout.setCenter(m.getKey());
        if(m.getValue()!=null)
            m.getValue().setMainApp(this.app);
        if(backScene!=currentScene){
           backScene=currentScene;
        }
        this.currentScene=scene;
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

/**
 * Abre una ventana modal
 * @param scene La ventana a abrir (ruta del archivo FXML)
 * @param title Título de la ventana modal.
 * @param parentController El controlador de la scene que abre el modal, this?
 * @param params Cualquier objeto que se quiere pasar como información de entrada al modal
 * @return Devuelve el controlador de la propia escena que se carga como modal
 */
public Controllers openModal(Scenes scene,String title,Controllers parentController,Object params){
    try {
        System.out.println(scene.getUrl());
        MapEntry<Parent, Controllers> m=loadFXML(scene.getUrl());
        Stage modalStage=new Stage();
        modalStage.setTitle(title);
        modalStage.initModality(Modality.APPLICATION_MODAL);
        modalStage.initOwner(this.app.mainStage);

        Scene modalScene=new Scene(m.getKey());
        modalStage.setScene(modalScene);

        if(m.getValue()!=null){
            m.getValue().setMainApp(this.app);
            ModalControllers mc=(ModalControllers)m.getValue();
            mc.setParentController(parentController);
            mc.setStage(modalStage);
            mc.setParams(params);
```

```
            }
            modalStage.showAndWait();
            return m.getValue();
        } catch (IOException ex) {
            ex.printStackTrace();
            return null;
        }
    }
    /**
     * No se usa en este proyecto, dado que nunca se cambiará el rootLayout.
     * Se deja como documentación
     * @param fxml
     * @throws IOException
     */
    private void setRoot(Scenes scene) throws IOException {
        this.backScene=null;
        this.currentScene=null;
        MapEntry<Parent, Controllers> m=loadFXML(scene.getUrl());
        this.app.scene.setRoot(m.getKey());
        this.app.controller=(AppController)m.getValue();
    }

    @FXML
    private MenuItem con;

    @FXML
    public void connectionsMenu(){
        changeScene(Scenes.CONN);
    }
    @FXML
    public void AboutPage(){
        changeScene(Scenes.ABOUT);
    }
    @FXML
    public void closeApp(){
        System.exit(0);
    }
    @FXML
    public void enableCon(){
        if(con!=null)
        con.setDisable(false);
    }
    @FXML
    public void disableCon(){
        if(con!=null)
        con.setDisable(true);
    }
    @FXML
    public void title(String txt){
        this.app.mainStage.setTitle(txt+"                                                                                                    (
"+(this.currentConnection==null?"Desconectado":this.currentConnection.getName())+" )");
    }

}
```

Se han incluido varias mejoras, entre las cuales están:

- Un método para abrir ventanas modales (openModal)
- Cargar la conexión por defecto de las preferencias del usuario cuando se carga el controlador por primera vez (onLoad)
- Actualizar el título de la ventana en función de si hay una conexión por defecto establecida o no (title).

*PrimaryController.java* no ha sido modificado. *primary.fxml* tampoco.

*ConnectionController.java*

```
package com.carlosserrano.proyectojavafx.controller;

import com.carlosserrano.proyectojavafx.model.Connection;
import com.carlosserrano.proyectojavafx.utils.Dialog;
import com.carlosserrano.proyectojavafx.utils.MapEntry;
import com.carlosserrano.proyectojavafx.utils.PreferencesUtil;
import com.carlosserrano.proyectojavafx.utils.XMLUtil;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.beans.property.SimpleObjectProperty;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Label;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.layout.HBox;

public class ConnectionsController extends Controllers implements Initializable {

    //OBSERVABLE <--------
    public ObservableList<Connection> conns;

    @FXML
    private TableView<Connection> connTable;
    @FXML
    private TableColumn<Connection, String> nameConn;

    @FXML
    private Label L_nameConn;
    @FXML
    public Label L_type;
    @FXML
    private Label L_server;
    @FXML
    private Label L_user;
    @FXML
    private Label L_pass;

    @FXML
    private HBox tools;

    @Override
```

```
    public void initialize(URL url, ResourceBundle rb) {
        this.conns=FXCollections.observableArrayList();
        this.conns.addAll(XMLUtil.loadDataXML());

        nameConn.setCellValueFactory(cellData -> {
            if(cellData.getValue().equals(app.controller.currentConnection)){
                return new SimpleObjectProperty<>(cellData.getValue().getName()+" ( C )");
            }else{
                return cellData.getValue().getN();
            }

        });
        showConnDetails(null);
        connTable.getSelectionModel().selectedItemProperty().addListener(
                (observable, oldValue, newValue) -> showConnDetails(newValue)
        );
        //Add observable
        connTable.setItems(conns);
    }

    private void showConnDetails(Connection c) {
        if (c != null) {
            tools.setDisable(false);
            L_nameConn.setText(c.getName());
            L_type.setText(c.getType());
            L_server.setText(c.getServer());
            L_user.setText(c.getUserName());
            L_pass.setText(c.getPassword());

        } else {
            tools.setDisable(true);
            resetForm();
        }
    }

    private void resetForm() {
        L_nameConn.setText("");
        L_type.setText("");
        L_server.setText("");
        L_user.setText("");
        L_pass.setText("");
    }

    @FXML
    public void back() {
        this.app.controller.changeScene(Scenes.PRIMARY);
    }

    void onLoad() {
        this.app.controller.disableCon();
        this.app.controller.title("CONEXIONES");
    }

    @FXML
    private void handleNewConnection() {
        editConnectionController cc = (editConnectionController) app.controller.openModal(Scenes.EDIT,
"Nueva Conexión",this,null);
```

```
    }

    @FXML
    private void handleConConnection(){
        Connection c = connTable.getSelectionModel().getSelectedItem();
        if (c == null) {
            Dialog.showWarning("Aviso", "No hay conexión seleccionada", "Seleccione una conexión antes de
pulsar connectar");
        } else {
            app.controller.currentConnection=c;
            app.controller.title("CONEXIONES");
            PreferencesUtil.setPreference(c.getName());
            conns.add(new Connection()); //refresh gui
        }
    }

    @FXML
    private void handleEditConnection() {
        Connection c = connTable.getSelectionModel().getSelectedItem();
        if (c == null) {
            Dialog.showWarning("Aviso", "No hay conexión para editar", "Seleccione una conexión antes de
pulsar editar");
        } else {
            editConnectionController cc = (editConnectionController) app.controller.openModal(Scenes.EDIT,
"Editando Conexión",this,c);
        }
    }

    @FXML
    private void handleRemoveConnection() {
        Connection c = connTable.getSelectionModel().getSelectedItem();
        if (c == null) {
            Dialog.showWarning("Aviso", "No hay conexión para borrar", "Seleccione una conexión antes de
pulsar borrar");
        } else {
            if(c.equals(app.controller.currentConnection)){
                app.controller.currentConnection=null;
                PreferencesUtil.setPreference(null);
                app.controller.title("CONEXIONES");
            }
            conns.remove(c);
            XMLUtil.writeDataXML(conns);
        }
    }

    @Override
    void doOnCloseModal(Object response) {
        if (response != null) {
            MapEntry<Connection, Boolean> r = (MapEntry<Connection,Boolean>) response;
            if(r.getValue()){  //new conn to be added
                conns.add(r.getKey());
            }
            showConnDetails(r.getKey());  //update GUI
            XMLUtil.writeDataXML(conns);
            //SAVE
        }
```

```
    }
}
```

Analicemos las diferentes partes de este archivo:

- public ObservableList<Connection> conns; Es un tipo de lista observable. Es decir, está relacionada con la programación reactiva. ¿Qué quiere decir? Que este tipo de variables "avisan" automáticamente cuando se ha realizado una modificación en su contenido. No debemos estar "preguntando" constantemente si se ha realizado o no una modificación. Esto es útil para diferentes contextos, pero muy interesante para GUIS (interfaces), de esta manera esta lista es la que "pintaremos" en una tabla. Cuando se realice un cambio en la lista, el Observable avisará a la tabla y la refrescará sin que nosotros tengamos que preocuparnos (casi).

  @FXML
- private TableView<Connection> connTable; Esta variable, al igual de las marcadas como @FXML está ligada (binded) con elementos visuales de la interfaz y nos permitirán acceder a estos elementos desde el controlador.

  Initialize, este método que se ejecuta cuando se carga la escena y su controlador realiza las siguientes operaciones:
  1. Crea la lista vacía.
  2. Carga la lista desde el archivo XML y la añade a la colección vacía que acabamos de crear.
  3. Define cómo añadir los datos a la tabla mediante su función setCellValueFactory:

     ```
     nameConn.setCellValueFactory(cellData -> {
     if(cellData.getValue().equals(app.controller.currentConnection)){
     return new SimpleObjectProperty<>(cellData.getValue().getName()+" ( C )");
     }else{
     return cellData.getValue().getN();
     }
     });
     ```

     Recibe una función lambda, por cada elemento que se inserte se ejecutará dicha función anónima que recibe como variable cellData que será el elemento iésimo iterado, es decir, cada una de las Connections que insertemos.

     Por cada Connection que insertemos compararemos si es igual a la que establecida como actual (la predefinida) y si es cierto el valor de la columna que rellenamos es el propio nombre de la conexión + "( C ) ", sino solo el nombre de la conexión.
  4. Define qué debe hacerse cuando un elemento de la tabla es seleccionado mediante su método getSelectionMode().selectedItemProperty.addListener:

     ```
     connTable.getSelectionModel().selectedItemProperty().addListener(
     (observable, oldValue, newValue) -> showConnDetails(newValue)
     );
     ```

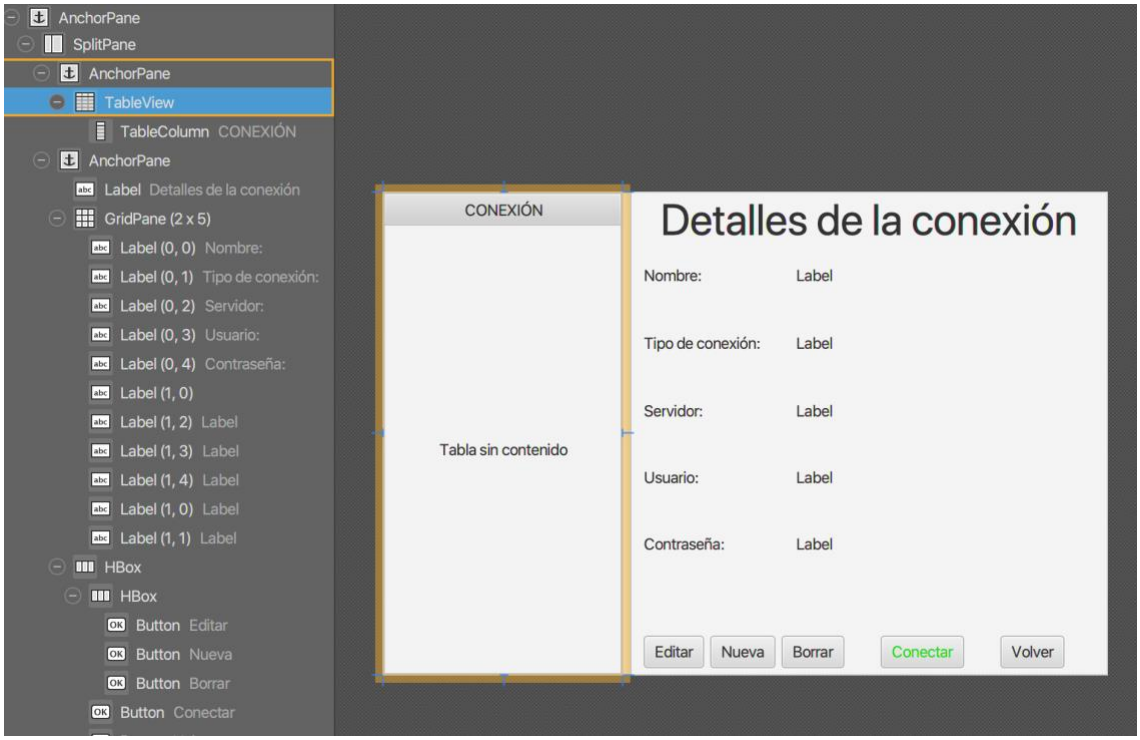
     De igual forma cada vez que se seleccione un ítem se ejecuta una función lambda, en este caso recibe tres entradas, el observable, el valor anterior (si hubiera cambiado) y el valor nuevo. Lo que realmente recibimos es la

conexión en sí relacionada con la entrada clicada. Esa conexión es pasada a showConnDetails que rellena etiquetas que nos permiten ver los detalles de dicha conexión. En breve veremos la pantalla y se comprenderá mejor su función.

- Los métodos que comienzan por handleXXX están relacionados con las acciones de los botones: abren modal para crear o editar, borra entrada o establece como conexión por defecto.

- doOnCloseModal es ejecutado cuando el modal es cerrado. ¿Qué modal? La pantalla para crear una nueva conexión o editarla. En función de lo que nos devuelva añadimos o no y actualizamos el archivo XML.

Veamos su pantalla *connections.fxml*




A la izquierda del SpliPane vertical tenemos un TableView (connTable) con una columna para el nombre de la conexión (CONEXIÓN). En esa tabla "pintaremos" nuestra lista Obserable conns.
A la derecha tenemos una rejilla (Grid) con etiquetas para ver los detalles de la conexión que seleccionemos en la tabla. En la parte inferior un HBox que nos permite organizar nuestra botonera.

El código es:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.carlosserrano.proyectojavafx.controller.ConnectionsController">
  <children>
    <SplitPane dividerPositions="0.29797979797979796" layoutX="190.0" layoutY="120.0" prefHeight="400.0" prefWidth="600.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
      <items>
       <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">
          <children>
            <TableView fx:id="connTable" layoutX="-13.0" layoutY="-19.0" prefHeight="398.0" prefWidth="174.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
              <columns>
                <TableColumn fx:id="nameConn" prefWidth="173.0" text="CONEXIONES" />
              </columns>
              <columnResizePolicy>
                <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
              </columnResizePolicy>
            </TableView>
          </children>
        </AnchorPane>
       <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">
          <children>
            <Label layoutX="14.0" layoutY="14.0" text="Detalles de la conexión" AnchorPane.leftAnchor="0.0" AnchorPane.topAnchor="0.0">
              <font>
                <Font size="26.0" />
              </font>
              <padding>
                <Insets left="10.0" top="5.0" />
              </padding>
            </Label>
            <GridPane layoutX="30.0" layoutY="39.0" prefHeight="317.0" prefWidth="416.0" AnchorPane.bottomAnchor="60.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="40.0">
              <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="204.0" minWidth="10.0" prefWidth="150.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="266.0" minWidth="10.0" prefWidth="266.0" />
              </columnConstraints>
              <rowConstraints>
```

```
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
              </rowConstraints>
              <children>
                <Label text="Nombre" />
                <Label text="Tipo" GridPane.rowIndex="1" />
                <Label text="Servidor" GridPane.rowIndex="2" />
                <Label text="Usuario" GridPane.rowIndex="3" />
                <Label text="Contraseña" GridPane.rowIndex="4" />
                <Label fx:id="L_nameConn" text="Label" GridPane.columnIndex="1" />
                <Label fx:id="L_type" text="Label" GridPane.columnIndex="1" GridPane.rowIndex="1" />
                <Label fx:id="L_server" text="Label" GridPane.columnIndex="1" GridPane.rowIndex="2" />
                <Label fx:id="L_user" text="Label" GridPane.columnIndex="1" GridPane.rowIndex="3" />
                <Label fx:id="L_pass" text="Label" GridPane.columnIndex="1" GridPane.rowIndex="4" />
              </children>
            </GridPane>
            <HBox fx:id="tools" layoutY="357.0" spacing="5.0" AnchorPane.bottomAnchor="0.0">
              <children>
                <Button                    layoutY="357.0"                    mnemonicParsing="false"
onAction="#handleRemoveConnection" text="Borrar" />
                <Button          layoutX="55.0"          layoutY="357.0"          mnemonicParsing="false"
onAction="#handleEditConnection" text="Editar" />
                <Button          layoutX="110.0"          layoutY="357.0"          mnemonicParsing="false"
onAction="#handleConConnection" text="Conectar" textFill="#45b500" />
              </children>
              <padding>
                <Insets bottom="2.0" left="2.0" />
              </padding>
            </HBox>
            <HBox  layoutX="286.0"  layoutY="371.0"  spacing="2.0"  AnchorPane.bottomAnchor="0.0"
AnchorPane.rightAnchor="0.0">
              <children>
                <Button          layoutX="286.0"          layoutY="350.0"          mnemonicParsing="false"
onAction="#handleNewConnection" text="Nueva" AnchorPane.bottomAnchor="0.0" />
                <Button  layoutX="341.0"  layoutY="350.0"  mnemonicParsing="false"  onAction="#back"
text="Volver" AnchorPane.bottomAnchor="0.0" />
              </children>
              <padding>
                <Insets bottom="2.0" right="2.0" />
              </padding>
            </HBox>
          </children>
        </AnchorPane>
      </items>
    </SplitPane>
  </children>
</AnchorPane>
```

Cuando pulsamos editar o nueva abriremos en modal la siguiente pantalla.
*editConnectionController.java*

```
package com.carlosserrano.proyectojavafx.controller;
```

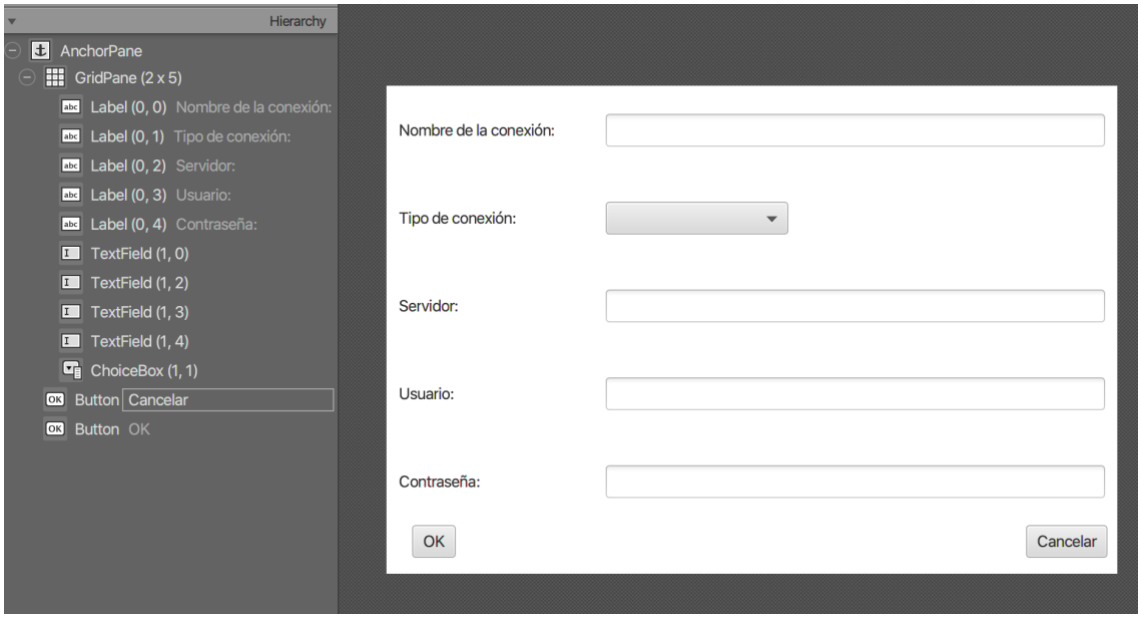
```
import com.carlosserrano.proyectojavafx.model.Connection;
import com.carlosserrano.proyectojavafx.model.ConnectionsType;
import com.carlosserrano.proyectojavafx.utils.Dialog;
import com.carlosserrano.proyectojavafx.utils.MapEntry;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class editConnectionController extends ModalControllers implements Initializable {

    private Connection c;

    private boolean creating;

    @FXML
    private TextField L_name;
    @FXML
    private TextField L_server;
    @FXML
    private TextField L_user;
    @FXML
    private TextField L_pass;

    @FXML
    public ChoiceBox<String> type;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        if (type != null) {
            for (ConnectionsType _type : Connection.connectionTypes.values()) {
                type.getItems().add(_type.getType());
            }
        }
    }

    public void setParams(Object p) {
        Connection c = (Connection) p;
        if (c == null) {
            this.c = new Connection();
            resetAll();
            creating = true;
        } else {
            this.c = c;
            L_name.setText(c.getName());
            L_server.setText(c.getServer());
            L_user.setText(c.getUserName());
            L_pass.setText(c.getPassword());
            type.getSelectionModel().select(c.getType());
            L_name.setEditable(false);
            creating = false;
        }
    }
```

```
public void setModalStage(Stage s) {
    this.stage = s;
}

@FXML
public void handleOK() {
    if (isValid()) {
        if (c == null) {
            this.c = new Connection();
            this.creating = true;
        }
        this.c.setName(L_name.getText());
        this.c.setType(type.getSelectionModel().getSelectedItem());
        this.c.setServer(L_server.getText());
        this.c.setUserName(L_user.getText());
        this.c.setPassword(L_pass.getText());

        if (this.parentController != null) {
            MapEntry<Connection, Boolean> response = new MapEntry<>(this.c, creating);
            this.parentController.doOnCloseModal(response);
        }

        this.stage.close();
    }
}

@FXML
public void handleCancel() {
    System.out.println(this.parentController);
    if (this.parentController != null) {
        this.parentController.doOnCloseModal(null);
    }
    this.stage.close();
}

private boolean isValid() {
    String error = "";

    if (L_name.getText() == null || L_name.getText().length() == 0) {
        error += " Nombre de conexión no válido \n";
    } else {
        if (creating) {
            //search a conn with same name
            Connection tmp = new Connection(L_name.getText());
            ConnectionsController cc = (ConnectionsController) parentController;
            if (cc.conns.contains(tmp)) {
                error += " Nombre de conexión no válido, ya existe una conexión con el mismo nombre\n";
            }
        }
    }

    if (error.length() == 0) {
        return true;
    } else {
        Dialog.showError("Error", "Corrija los errores", error);
        return false;
    }
```

```
    }

    private void resetAll() {
        type.getSelectionModel().select(1);
        L_name.setText("");
        L_server.setText("");
        L_user.setText("");
        L_pass.setText("");
    }
}
```

En el método initialize se recorre el enum Connection.connectionTypes para añadir las conexiones disponibles en un combobox (select) de la interfaz.
Mediante setParams se recibe la conexión a editar o null en caso de crear una nueva.
handleOk valida el formulario (comprueba que el nombre no esté repetido) y deuelve un mapEntry con la conexión y un boleano (true si es crear y false si es editar) a la pantalla de conexiones para que realice las operaciones pertinentes.

Su pantalla es *editConnection.fxml*




Y su código:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.carlosserrano.proyectojavafx.controller.editConnectionController">
  <children>
```

```
    <GridPane layoutX="43.0" layoutY="116.0" prefHeight="400.0" prefWidth="600.0"
AnchorPane.bottomAnchor="40.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
      <columnConstraints>
       <ColumnConstraints hgrow="SOMETIMES" maxWidth="296.0" minWidth="10.0"
prefWidth="180.0" />
       <ColumnConstraints hgrow="SOMETIMES" maxWidth="422.0" minWidth="10.0"
prefWidth="420.0" />
      </columnConstraints>
      <rowConstraints>
        <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
       <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
       <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
       <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
      </rowConstraints>
      <children>
        <Label text="Nombre de la conexión:" />
        <Label text="Tipo de conexión: " GridPane.rowIndex="1" />
        <Label text="Servidor:" GridPane.rowIndex="2" />
        <Label text="Usuario:" GridPane.rowIndex="3" />
        <Label text="Contraseña: " GridPane.rowIndex="4" />
        <TextField fx:id="L_name" GridPane.columnIndex="1" />
        <TextField fx:id="L_server" GridPane.columnIndex="1" GridPane.rowIndex="2" />
        <TextField fx:id="L_user" GridPane.columnIndex="1" GridPane.rowIndex="3" />
        <TextField fx:id="L_pass" GridPane.columnIndex="1" GridPane.rowIndex="4" />
        <ChoiceBox fx:id="type" prefWidth="150.0" GridPane.columnIndex="1" GridPane.rowIndex="1"
/>
      </children>
      <padding>
        <Insets left="10.0" right="10.0" />
      </padding>
    </GridPane>
    <Button layoutX="525.0" layoutY="360.0" mnemonicParsing="false" onAction="#handleCancel"
text="Cancelar" />
    <Button layoutX="21.0" layoutY="360.0" mnemonicParsing="false" onAction="#handleOK" text="OK"
/>
  </children>
</AnchorPane>
```

El proyecto completo se encuentra en el repositorio:
https://github.com/Developodo/JavaFX
(com.carlosserrano_ProyectoJavaFX_P2)

# 6 PARTE 3: JAVAFX y MYSQL

En esta sección desarrollaremos un CRUD simple con MYSQL, además introduciremos el uso de H2, una implementación para Java de bases relacionales embebidas (no necesitas un servicio ddbb, se crea en archivos locales).
Para ello, en este ejemplo, haremos uso del conector JDBC y sin mapeo Objeto-Relación, implementaremos las consultas. En temas más avanzados veremos el concepto de ORM.

Añadimos, por tanto, al archivo pom.xml las siguientes dependencias:

```
    </dependency>
        <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.19</version>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <version>1.4.200</version>
    </dependency>
```

Y en el archivo module-info.java debemos permitir su acceso:

```
module com.carlosserrano.proyectojavafx {
    requires javafx.controls;
    requires javafx.fxml;
    requires java.base;
    requires java.xml.bind;
    requires java.prefs;
    requires java.sql;
    requires com.h2database;

    opens com.carlosserrano.proyectojavafx.utils to java.xml.bind; //Para que JAXB pueda ejecutarse en XMLUtil
    opens com.carlosserrano.proyectojavafx.controller to javafx.fxml;
    opens com.carlosserrano.proyectojavafx.model to java.xml.bind; //Para que JAXB pueda ejecutarse en ConnectionWrapper

    exports com.carlosserrano.proyectojavafx;
    exports com.carlosserrano.proyectojavafx.model;  //para que JAXB pueda acceder a Connection y Connection wrapper
}
```

Comenzamos por el modelo, es decir aquellas clases que nos permiten realizar un mapeo directo de nuestros datos y el sistema. Nuestro CRUD se basará en el siguiente modelo E-R:


Es decir, tendremos una lista de contactos (con un id como clave). Dichas contactos, además del nombre y la fecha de nacimiento, podrán tener 0 o más canales de comunicación (relación 1:N). Dichos canales tienen un id, un tipo de canal (por ejemplo: teléfono) y un valor (por ejemplo: 555-55555).

Nuestra aplicación va a tener una base de datos llamada agenda con estas dos tablas. Los comandos SQL para generarla serían:

```
CREATE DATABASE IF NOT EXISTS `agenda` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `agenda`;

CREATE TABLE IF NOT EXISTS `channel` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `type` text NOT NULL,
  `value` text NOT NULL,
  `id_contact` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `CR` (`id_contact`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `contactos` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` text NOT NULL,
  `birthdate` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `channel`
  ADD CONSTRAINT `CR` FOREIGN KEY (`id_contact`) REFERENCES `contactos` (`id`);
```

Para modelar estos datos hemos creado en el paquete *com.carlosserranoproyectojavafx.model* dos clases:

*Channel.java*

```
package com.carlosserrano.proyectojavafx.model;

public class Channel {
   protected int id;
```

```
    protected String type;
    protected String value;
    protected int id_contact;

    public Channel(int id, String type, String value, int id_contact) {
        this.id = id;
        this.type = type;
        this.value = value;
        this.id_contact = id_contact;
    }

    public Channel() {
        this.id=-1;
        this.type="";
        this.value="";
        this.id_contact=-1;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public int getId_contact() {
        return id_contact;
    }

    public void setId_contact(int id_contact) {
        this.id_contact = id_contact;
    }

    @Override
    public String toString() {
        return "Channel{" + "id=" + id + ", type=" + type + ", value=" + value + ", id_parent=" + id_contact +
'}';
    }
    @Override
    public int hashCode() {
```

```
        int hash = 7;
        return hash;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Channel other = (Channel) obj;
        if (this.id != other.id) {
            return false;
        }
        return true;
    }
}
```

Como se observa es una clase convencional. Como único aspecto a tener en cuenta es que consideramos dos canales iguales si sus ids son los mismos.

### *Contact.java*

```
package com.carlosserrano.proyectojavafx.model;

import java.time.LocalDate;
import java.util.List;


public class Contact {
    protected int id;
    protected String nickname;
    protected List<Channel> channels;
    protected LocalDate birthDate;

    public Contact(int id, String nickname, List<Channel> channels, LocalDate birthDate) {
        this.id = id;
        this.nickname = nickname;
        this.channels = channels;
        this.birthDate = birthDate;
    }

    public Contact() {
        this(-1,"",null,LocalDate.now());
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
    public String getNickname() {
        return nickname;
    }

    public void setNickname(String nickname) {
        this.nickname = nickname;
    }

    public List<Channel> getChannels() {
        return channels;
    }

    public void setChannels(List<Channel> channels) {
        this.channels = channels;
    }

    public LocalDate getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(LocalDate birthDate) {
        this.birthDate = birthDate;
    }

    @Override
    public int hashCode() {
        int hash = 7;
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Contact other = (Contact) obj;
        if (this.id != other.id) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Contact{" + "id=" + id + ", nickname=" + nickname + ", emails=" + channels + ", birthDate=" +
birthDate + '}';
    }

}
```

Similar a la clase anterior, pero un cambio importante. En nuestro modelo de datos orientado a objetos, las relaciones siempre son atributos, en este caso los contactos tendrán un arrayList con los canales.
Imaginemos una relación N:M, en SQL tendríamos 3 tablas:

- Contacto
- Canal
- Contacto-Canal

Sin embargo, en el modelo orientado a objetos solo basta con dos clases:

- Contacto: tendrá un List con todos sus canales
- Canal: tendrá un List con todos los contactos que tienen dicho canal.

Es importante señalar que hemos definido como dueña de la relación a Contacto, es decir, en la clase Contact hay un puntero directo a sus canales, sin embargo en Channel no existe ninguna relación explícita, solo un entero con el id de su contacto, pero no puntero directo. Podríamos haberla hecho bidireccional, es decir en Channel habría un atributo:

```
Contact mi_contacto;
```

No obstante, por simplicidad lo dejaremos así.

Para gestionar todas las conexiones que realizaremos empleando el Driver JDBC, hemos creado en com.carlos.serrano.proyectojavafx.utils una clase que nos facilite el trabajo reutilizable:

### *ConnectionUtil.java*

```
package com.carlosserrano.proyectojavafx.utils;

import com.carlosserrano.proyectojavafx.controller.AppController;
import com.carlosserrano.proyectojavafx.model.Connection;
import com.carlosserrano.proyectojavafx.model.ConnectionsType;
import java.sql.Array;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class ConnectionUtil {

   public static java.sql.Connection connect(Connection c) {
      java.sql.Connection con = null;
      if (c == null) {
         return null;
      }
      try {
         String driver = c.getType();
         if (driver.equals("mySQL")) {
```

```
            con = DriverManager.getConnection("jdbc:mysql://" + c.getServer() +
"/agenda?useLegacyDatetimeCode=false&serverTimezone=UTC", c.getUserName(), c.getPassword());
        } else if (driver.equals("H2")) {
            //~/test
            try{
               Class.forName("org.h2.Driver");
             }catch(Exception e){}

            con = DriverManager.getConnection("jdbc:h2:"+c.getServer()+"/agenda");
        } else {
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://" + c.getServer() +
"/agenda?useLegacyDatetimeCode=false&serverTimezone=UTC", c.getUserName(), c.getPassword());
        }

        checkStructure(con);

      } catch (ClassNotFoundException | SQLException ex) {
         Dialog.showError("ERROR", "ERROR conectando con DDBB", ex.toString());
      }
      return con;
   }

   public static ResultSet execQuery(java.sql.Connection con, String q, List<Object> params) throws
SQLException {
      ResultSet result = null;
      if (con == null) {
         return null;
      }

      PreparedStatement ps = prepareQuery(con, q, params);
      result = ps.executeQuery();

      return result;
   }

   public static ResultSet execQuery(java.sql.Connection con, String q, Object param) throws
SQLException {
      List<Object> params = new ArrayList<>();
      params.add(param);
      return execQuery(con, q, params);
   }

   public static int execUpdate(java.sql.Connection con, String q, List<Object> params, boolean insert)
throws SQLException {
      if (con == null) {
         return -1;
      }

      PreparedStatement ps = prepareQuery(con, q, params);
      int result = ps.executeUpdate();
      //check if insert
      if (insert) {
         try (ResultSet generatedKeys = ps.getGeneratedKeys()) {
            if (generatedKeys.next()) {
               return generatedKeys.getInt(1);  //<-- return last id inserted
            } else {
```

```
                return -1;
            }
          }
        } else {
          return result;
        }

    }

    public static int execUpdate(java.sql.Connection con, String q, Object param, boolean insert) throws SQLException {
        List<Object> params = new ArrayList<>();
        params.add(param);
        return execUpdate(con, q, params, insert);
    }

    /**
     *
     */
    public static int is(Integer n) {
        return 0;
    }
    public static int is(Float n) {
        return 1;
    }
    public static int is(Double n) {
        return 2;
    }
    public static int is(Boolean n) {
        return 3;
    }
    public static int is(String n) {
        return 4;
    }
    public static int is(Array n){
        return 5;
    }
    public static int is(Object n) {
        return 6;
    }

    public static PreparedStatement prepareQuery(java.sql.Connection con, String q, List params) throws SQLException {
        PreparedStatement ps = null;
        ps = con.prepareStatement(q, Statement.RETURN_GENERATED_KEYS); //<-- solo para insert es útil
        if (params != null) {
            int i = 1;
            for (Object o : params) {
                switch (is(params)) {
                    case 0:
                        ps.setInt(i++, (Integer) o);
                        break;
                    case 1:
                        ps.setFloat(i++, (Float) o);
                        break;
                    case 2:
                        ps.setDouble(i++, (Double) o);
```

```
                    break;
                case 3:
                    ps.setBoolean(i++, (Boolean) o);
                    break;
                case 4:
                    ps.setString(i++, (String) o);
                    break;
                case 5:
                    ps.setArray(i++, (Array) o);
                    break;
                default:
                    ps.setObject(i++, o);
            }
        }
    }
    return ps;
  }

  public static void checkStructure(java.sql.Connection con) {
    try {
      String sql1,sql2;

      if (AppController.currentConnection.getType().equals(ConnectionsType.MYSQL.getType())) {
        sql1 = "CREATE TABLE IF NOT EXISTS `contactos` ("
          + "  `id` int(11) NOT NULL AUTO_INCREMENT,"
          + "  `name` text NOT NULL,"
          + "  `birthdate` datetime NOT NULL,"
          + "  PRIMARY KEY (`id`)"
          + ")";
         sql2= "CREATE TABLE IF NOT EXISTS `channel` ("
          + "  `id` int(11) NOT NULL AUTO_INCREMENT,"
          + "  `type` text NOT NULL,"
          + "  `value` text NOT NULL,"
          + "  `id_contact` int(11) NOT NULL,"
          + "  PRIMARY KEY (`id`)"
          + ")";
        // sql3 = "ALTER TABLE `channel`"
        //       + "  ADD CONSTRAINT `CR` FOREIGN KEY (`id_contact`) REFERENCES `contactos` (`id`);";

      } else {
        sql1 = "CREATE TABLE IF NOT EXISTS contactos (id INT PRIMARY KEY auto_increment, name
VARCHAR(255), birthdate VARCHAR(255));";
        sql2 = "CREATE TABLE IF NOT EXISTS channel (id INT PRIMARY KEY auto_increment, type
VARCHAR(255), value VARCHAR(255), id_contact INT );";
      }
      con.setAutoCommit(false);
      ConnectionUtil.execUpdate(con, sql1, null, false);
      ConnectionUtil.execUpdate(con, sql2, null, false);
      con.commit();
      con.setAutoCommit(true);

    } catch (SQLException ex) {
      Dialog.showError("ERROR", "Error creando tablas", ex.toString());
    }
  }
}
```

Merece la pena analizar el código puesto que existen muchos conceptos interesantes

El método public static java.sql.Connection connect(Connection c):
Recibe una Connection (con los datos) y devuelve una conexión sql. Dependiendo si usamos MySQL o H2, la creamos con su driver adecuado. Como se puede observar en caso de mysql se fuerza a que la base de datos se denomine agenda (luego debe estar creada). En el caso de H2, se configura la ruta, pero de igual forma, se fuerza a que el archivo (con lo cual la base de datos) se denomine agenda. El método llama a checkStructure que crea las tablas si no existieran (el mecanismo diverge un poco si es Mysql o H2).

En checkStructure se emplea el siguiente código:

```
con.setAutoCommit(false);
 ConnectionUtil.execUpdate(con, sql1, null, false);
 ConnectionUtil.execUpdate(con, sql2, null, false);
 con.commit();
 con.setAutoCommit(true);
```

Este código permite realizar una transacción. Es decir que

```
ConnectionUtil.execUpdate(con, sql1, null, false);
 ConnectionUtil.execUpdate(con, sql2, null, false);
```

Se ejecutarán de forma atómica, se ocurriese, por ejemplo, un error en la ejecución de sql2, se realizaría un Rollback, es decir, la base de datos volvería a su estado original, antes de ejecutar sql1. Esto lo empleamos cuando tenemos operaciones que dependen una de otras en integridad y queremos asegurarnos de que o se ejecutan todas o ninguna.

Existen exeQuery y execUpdate, ambos sobrecargado, puesto que o reciben una lista de paramétros o solo un parámetro (por simplificar, por ejemplo cuando solo se tiene que pasar un id y no deseamos crear un List solo para esto). La diferencia entre Query y Update es que el primero se emplea para operaciones SELECT y el segundo para INSERT, UPDATE y REMOVE. Además, este último recibe un parámetro insert (boolean) que le indica en caso de INSERT que devuelva el último id insertado, en caso negativo devuelve el número de filas afectadas por el UPDATE o DELETE.

Como se verá más adelante, hemos empleado PreparedStatement, por ello hemos creado un método prepareQuery que recorre la lista de parámetros y los va “insertando” en la query. Para diferenciar si insrta un Int de un Float, por ejemplo, se emplea un método sobrecargado is que nos permite diferenciar los tipos fundamentales para preparar de forma más automática la query.

Y ahora entramos en el apartado DAO, para ello hemos creado un paquete dentro de model: *com.carlosserrano.proyectojavafx.model.dao*.

¿Qué es DAO? Data Access Object. Es decir, un objeto que permite el acceso a datos. ¿Qué entendemos por acceso a datos? La conexión de nuestro sistema con aquel que almacena los datos de forma persistente: MySQL, H2, un servicio…

Hemos creado Channel y Contact, dichas clases serán los objetos que almacenen nuestros datos, pero como pasamos de un modelo E-R a un modelo OO (Orientado a Objetos). Con lo que se denomina Mapeo Objeto Relación (ORM). Para ello, en temas más avanzados hablaremos de frameworks que nos ayudan a realizarlo de forma automática, como puede ser JPA.
En este ejemplo, lo haremos de forma artesanal, dado lo enriquecedor que será comprobar empíricamente las dificultades que supone la denominada impedancia relación objeto, es decir, el mapeo de datos entre una base de datos relacional y nuestro sistema orientado a objetos no es trivial.

Por tanto, para ello, vamos a crear objetos que contengan toda la información, pero que además posea la capacidad de mapear sus datos con el modelo relacional, es decir existirá una clase con el esqueleto de Channel, pero con la habilidad de conectarse a la base de datos y realizar solita los cambios oportunos en la tabla cuando en la instancia ocurran. Por ejemplo, si tengo esta situación:

```
Channel c;  //con id=1, tipo=teléfono y valor=555-5555
```

Y en la aplicación ocurre esto:

```
c.setValue(“888-8888)
```

Esta clase solita debería ejecutar un:

```
UPDATE channel SET value=”888-8888” WHERE id=1
```

Este tipo de clases se denominan DAO. He creado una interfaz DAO para obligar a que compartan ciertas características de diseño.

### *Dao.class*

```
package com.carlosserrano.proyectojavafx.model.dao;

public interface Dao {

   public default String test(){
      return "hola";
  }

   /**
    * set boolean persist a true
    * Cualquier cambio en la instancia desencadena una acutalización automática de la tabla
    */
  void persist();
  /**
   * set boolean persist a false
   * Los cambios en la instancia no actualizan la tabla automáticamente. Se
   * requiere una llamada a save explícita para que se actualice la tabla
   */
  void detach();
  /**
   * Elimina en la tabla el elemento que coincida con esta instancia (mismo id)
```

```
    */
    void remove();
    /**
     * En caso de que id=-1 realiza un INSERT
     * En caso de que id>0 realiza un UPDATE
     * Estableciendo en la tabla los valores corrrespondientes a esta instancia
     */
    void save();

    /**
     * Ojo: tanto en save como remove, hay que tener en cuenta los cambios que ocurran
     * en las relaciones.
     */

}
```

Tendrán dos métodos remove y save que permitirán borra, insertar y actualizar los datos. Le otorgamos una funcionalidad interesenta mediante persist y detach. En caso de que persist esté a true (existirá una variable con dicho nombre) cualquier ejecución de un setter no lo cambiará el valor de la instancia, sino que modificará la tabla. En caso de que no queramos sobrecargar la app con peticiones SQL encadenadas por muchas llamadas a setter seguidas, podemos con detach establecer persis a false. En ese caso, las llamadas a setter solo actuarán de forma convencional, accediendo al atributo de la instancia. Por defecto persist será false.

He creado el método test

```
public default String test(){
      return "hola";
  }
```

No tiene ninguna utilidad, pero para recordar que con el modificador default una instancia, a partir de la JDK8, puede contener métodos con cuerpo. Puede ser interesante en algunas situaciones.

*ChannelDAO.java*

```
package com.carlosserrano.proyectojavafx.model.dao;

import com.carlosserrano.proyectojavafx.controller.AppController;
import com.carlosserrano.proyectojavafx.model.Channel;
import com.carlosserrano.proyectojavafx.utils.ConnectionUtil;
import com.carlosserrano.proyectojavafx.utils.Dialog;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class ChannelDao extends Channel {

    enum queries {
        INSERT("INSERT INTO channel (id,type,value,id_contact) VALUES (NULL,?,?,?)"),
        ALL("SELECT * FROM channel"),
```

```
    GETBYID("SELECT * FROM channel WHERE id=?"),
    FINDBYCONTACT("SELECT * FROM channel WHERE id_contact =?"),
    FINDBYVALUE("SELECT * FROM channel WHERE value LIKE ?"),
    UPDATE("UPDATE channel SET type = ?, value = ? WHERE id = ?"),
    REMOVE("DELETE FROM channel WHERE id=?");
    private String q;

    queries(String q) {
        this.q = q;
    }

    public String getQ() {
        return this.q;
    }
}

Connection con;
private boolean persist;

public ChannelDao(int id, String type, String value, int id_contact) {
    super(id, type, value, id_contact);
    con = ConnectionUtil.connect(AppController.currentConnection);
    persist = false;
}

public ChannelDao() {
    super();
    con = ConnectionUtil.connect(AppController.currentConnection);
    persist = false;
}

public ChannelDao(Channel c) {
    this(c.getId(), c.getType(), c.getValue(), c.getId_contact());
}

public ChannelDao(int i) {
    this();
    try {
        ResultSet rs = ConnectionUtil.execQuery(con, ContactDao.queries.GETBYID.getQ(), i);

        if (rs != null) {

            while (rs.next()) {
                Channel c = instanceBuilder(rs);
                this.id = c.getId();
                this.type = c.getType();
                this.value = c.getValue();
                this.id_contact = c.getId_contact();
            }

        }
    } catch (SQLException ex) {
        Dialog.showError("ERRPR", "Error cargando el canal", ex.toString());
    }
}

public void persist() {
```

```
    this.persist = true;
}

public void detach() {
    this.persist = false;
}

@Override
public void setId_contact(int id_contact) {
    super.setId_contact(id_contact);
}

@Override
public void setValue(String value) {
    super.setValue(value);
    if (persist) {
        save();
    }
}

@Override
public void setType(String type) {
    super.setType(type); //To change body of generated methods, choose Tools | Templates.
    if (persist) {
        save();
    }
}

@Override
public void setId(int id) {
    //not allowed
}

public void save() {
    queries q;
    List<Object> params = new ArrayList<>();
    params.add(this.getType());
    params.add(this.getValue());

    if (this.id == -1) {
        q = queries.INSERT;
        params.add(this.getId_contact());
    } else {
        q = queries.UPDATE;
        params.add(this.id);
    }

    try {
        int rs = ConnectionUtil.execUpdate(con, q.getQ(), params,(q==queries.INSERT?true:false));
        if ( q == queries.INSERT) {
            this.id=rs;
        }
    } catch (SQLException ex) {
        Dialog.showError("ERROR", "Error guardando canal", ex.toString());
    }

}
```

```
public void remove() {
    if (this.id != -1) {
        try {
            int rs = ConnectionUtil.execUpdate(con, queries.REMOVE.getQ(), this.id,false);
        } catch (SQLException ex) {
            Dialog.showError("ERROR", "Error borrando contacto", ex.toString());
        }
    }
}

public static Channel instanceBuilder(ResultSet rs) {

    Channel c = new Channel();
    if (rs != null) {
        try {
            c.setId(rs.getInt("id"));
            c.setType(rs.getString("type"));
            c.setValue(rs.getString("value"));
            c.setId_contact(rs.getInt("id_contact"));
            //falta lazy contacts
        } catch (SQLException ex) {
            Dialog.showError("Error SQL", "SQL creando contacto", ex.toString());
        }

    }
    return c;
}

public static List<Channel> getAll(Connection con) {
    List<Channel> result = new ArrayList<>();
    try {
        ResultSet rs = ConnectionUtil.execQuery(con, queries.ALL.getQ(), null);
        if (rs != null) {
            while (rs.next()) {
                Channel n = ChannelDao.instanceBuilder(rs);
                result.add(n);
            }
        }
    } catch (SQLException ex) {
        Dialog.showError("ERROR", "Error cargando el contactos", ex.toString());
    }
    return result;
}

public static List<Channel> getByValue(Connection con, String value) {
    List<Channel> result = new ArrayList<>();
    try {
        ResultSet rs = ConnectionUtil.execQuery(con, queries.FINDBYVALUE.getQ(), "%" +value + "%");
        if (rs != null) {
            while (rs.next()) {
                Channel n = ChannelDao.instanceBuilder(rs);
                result.add(n);
            }
        }
    } catch (SQLException ex) {
        Dialog.showError("ERROR", "Error cargando el canal", ex.toString());
```

```
        }
        return result;
    }

    public static List<Channel> getByContact(Connection con, int id_contact) {
        List<Channel> result = new ArrayList<>();
        try {
            ResultSet rs = ConnectionUtil.execQuery(con, queries.FINDBYCONTACT.getQ(), id_contact);
            if (rs != null) {
                while (rs.next()) {
                    Channel n = ChannelDao.instanceBuilder(rs);
                    result.add(n);
                }
            }
        } catch (SQLException ex) {
            Dialog.showError("ERROR", "Error cargando el canal", ex.toString());
        }
        return result;
    }
}
```

Hereda de Channel, puesto que debe tener su estructura de datos e implementa Dao.

Como se observa contiene un enum con las consultas que se definen sobre el objeto (tantas como queramos ponerlas). A dichas consultas existen métodos para realizarlas.
Es importante que veamos cómo están sobreescritos los setter para persistir de forma automática y como está implementado save y remove.

Tiene algunos métodos estáticos que nos sirven de utilidad, como:
instanceBuilder que crea una instancia de Channel a partir de un resultado de consulta.
getAll que devuelve todos los canales de la base de datos en forma de lista
getByValue que deuvelve solo aquellos cuyo valor sea similar al pasado como parámetro.
getByContact que devuelve una lista de todos los canales de un contacto dado (su id).

Por último, existe un constructor muy interesante que es aquel el cual recibe un id y automáticamente carga en la instancia los valores del elemento de la tabla que coincida con dicho id (si existe).

Como vemos ChannelDAO nos permite realizar cualquier operación con la tabla channel, creando objetos Channel mapeados de los elementos de dicha tabla.
Sin embargo, ContactDAO no es tan senciila, puesto que es la dueña de la relación y contiene en su interior Channel. Por lo que, como veremos, hay que tener muchas consideraciones en cuenta.

*ContactDAO.java*

```
package com.carlosserrano.proyectojavafx.model.dao;

import com.carlosserrano.proyectojavafx.controller.AppController;
import com.carlosserrano.proyectojavafx.model.Channel;
import com.carlosserrano.proyectojavafx.model.Contact;
import com.carlosserrano.proyectojavafx.utils.ConnectionUtil;
import com.carlosserrano.proyectojavafx.utils.Dialog;
```

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

public class ContactDao extends Contact implements Dao{

   enum queries {
      INSERT("INSERT INTO contactos (id,name,birthdate) VALUES (NULL,?,?)"),
      ALL("SELECT * FROM contactos"),
      GETCHANNELS("SELECT * FROM channel WHERE id_contact=?"),
      GETBYID("SELECT * FROM contactos WHERE id=?"),
      FINDBYID("SELECT * FROM contactos WHERE id IN "), //<-- ojo con esta, hay formas más elegantes
      FINDBYNAME("SELECT * FROM contactos WHERE name LIKE ?"),
      UPDATE("UPDATE contactos SET name = ?, birthdate = ? WHERE id = ?"),
      REMOVE("DELETE FROM contactos WHERE id=?");
      private String q;

      queries(String q) {
         this.q = q;
      }

      public String getQ() {
         return this.q;
      }
   }

   Connection con;
   private boolean persist;

   public ContactDao(int id, String nickname, List<Channel> channels, LocalDate birthDate) {
      super(id, nickname, channels, birthDate);
      con = ConnectionUtil.connect(AppController.currentConnection);
      persist = false;
   }

   public ContactDao() {
      super();
      con = ConnectionUtil.connect(AppController.currentConnection);
      persist = false;
   }

   //DAO
   public ContactDao(Contact c) {
      this(c.getId(), c.getNickname(), c.getChannels(), c.getBirthDate());
   }

   public ContactDao(int i) {
      this();
      List<Object> params = new ArrayList<>();
      params.add(i);
      try {
         ResultSet rs = ConnectionUtil.execQuery(con, queries.GETBYID.getQ(), params);

         if (rs != null) {
```

```
            while (rs.next()) {
                Contact c = instanceBuilder(rs);
                this.id = c.getId();
                this.nickname = c.getNickname();
                this.birthDate = c.getBirthDate();
                this.channels = c.getChannels();
            }

        }
    } catch (SQLException ex) {
        Dialog.showError("ERRPR", "Error cargando el contacto", ex.toString());
    }
}

public void persist() {
    this.persist = true;
}

public void detach() {
    this.persist = false;
}

@Override
public List<Channel> getChannels() {
    if (this.channels == null) {
        //lazy retrieve
        List<Channel> result = ChannelDao.getByContact(con, id);

        this.setChannels(result);
    }
    return super.getChannels(); //To change body of generated methods, choose Tools | Templates.
}

@Override
public void setBirthDate(LocalDate birthDate) {
    super.setBirthDate(birthDate); //To change body of generated methods, choose Tools | Templates.
    if (persist) {
        save();
    }
}

@Override
public void setChannels(List<Channel> channels) {
    super.setChannels(channels); //To change body of generated methods, choose Tools | Templates.
    if (persist) {
        save();
    }
}

@Override
public void setNickname(String nickname) {
    super.setNickname(nickname); //To change body of generated methods, choose Tools | Templates.
    if (persist) {
        save();
    }
}
```

```
@Override
public void setId(int id) {
    /*super.setId(id); //To change body of generated methods, choose Tools | Templates.
    if(persist){
        save();
    }*/
    //primary key cannot be changed
}

public void save() {
    queries q;
    List<Object> params = new ArrayList<>();
    params.add(this.getNickname());
    params.add(this.getBirthDate());

    if (this.id == -1) {
        q = queries.INSERT;
    } else {
        q = queries.UPDATE;
        params.add(this.id);
    }

    try {
        //Comienza transacción
        con.setAutoCommit(false);
        if (channels != null) {  //si se ha modificado algo sobre channels
            //Boorando aquellos que no están ya -> coherencia
            List<Channel> oldChannels = ChannelDao.getByContact(con, id);
            for (Channel oldChannel : oldChannels) {
                if (!channels.contains(oldChannel)) {
                    ChannelDao cd = new ChannelDao(oldChannel);
                    cd.remove();
                }
            }

            //Actualizando o insertando los nuevos
            for (Channel newChannels : channels) {
                ChannelDao cd = new ChannelDao(newChannels);
                cd.setId_contact(id);  //me aseguro de la relación
                cd.save();
            }
        }

        int rs = ConnectionUtil.execUpdate(con, q.getQ(), params,(q==queries.INSERT?true:false));
        if ( q == ContactDao.queries.INSERT) {
            this.id=rs;
        }
        //Fin de la transacción
        con.commit();
        con.setAutoCommit(true);
    } catch (SQLException ex) {
        Dialog.showError("ERROR", "Error guardando contacto", ex.toString());
    }

}
```

```
public void remove() {
    if (this.id != -1) {
        try {
            //Comienza transacción
            con.setAutoCommit(false);

            //Boorando aquellos que no están ya -> coherencia
            List<Channel> oldChannels = ChannelDao.getByContact(con, id);

            for (Channel oldChannel : oldChannels) {
                ChannelDao cd = new ChannelDao(oldChannel);
                cd.remove();
            }
            int rs = ConnectionUtil.execUpdate(con, queries.REMOVE.getQ(), this.id,false);

            //Fin de la transacción
            con.commit();
            con.setAutoCommit(true);

        } catch (SQLException ex) {
            Dialog.showError("ERROR", "Error borrando contacto", ex.toString());
        }
    }
}

// UTILS for CONTACT DAO

public static Contact instanceBuilder(ResultSet rs) {
    //ojo rs.getMetaData()
    Contact c = new Contact();
    if (rs != null) {
        try {
            c.setId(rs.getInt("id"));
            c.setNickname(rs.getString("name"));
            c.setBirthDate(rs.getDate("birthdate").toLocalDate());
            //falta lazy contacts
        } catch (SQLException ex) {
            Dialog.showError("Error SQL", "SQL creando contacto", ex.toString());
        }

    }
    return c;
}

public static List<Contact> getAll(Connection con) {
    List<Contact> result = new ArrayList<>();
    try {
        ResultSet rs = ConnectionUtil.execQuery(con, queries.ALL.getQ(), null);
        if (rs != null) {
            while (rs.next()) {
                Contact n = ContactDao.instanceBuilder(rs);
                result.add(n);
            }
        }
    } catch (SQLException ex) {
        Dialog.showError("ERRPR", "Error cargando el contactos", ex.toString());
    }
```

```
        return result;
    }

    public static List<Contact> getByName(Connection con, String name) {
        List<Contact> result = new ArrayList<>();
        try {
            ResultSet rs = ConnectionUtil.execQuery(con, queries.FINDBYNAME.getQ(), name+"%");
            if (rs != null) {
                while (rs.next()) {
                    Contact n = ContactDao.instanceBuilder(rs);
                    result.add(n);
                }
            }
        } catch (SQLException ex) {
            Dialog.showError("ERRPR", "Error cargando el contactos", ex.toString());
        }
        return result;
    }

    public static List<Contact> getById(Connection con, List<Integer> ids) {
        List<Contact> result = new ArrayList<>();
        try {
            List<String> newList = new ArrayList<String>(ids.size());
            for (Integer myInt : ids) {
                newList.add(String.valueOf(myInt));
            }
            String queryTotal=queries.FINDBYID.getQ()+"("+String.join(",", newList)+");";

            ResultSet rs = ConnectionUtil.execQuery(con, queryTotal, null) ;
            if (rs != null) {
                while (rs.next()) {
                    Contact n = ContactDao.instanceBuilder(rs);
                    result.add(n);
                }
            }
        } catch (SQLException ex) {
            Dialog.showError("ERRPR", "Error cargando el contactos", ex.toString());
        }
        return result;
    }

}
```

La estructura es similar a la de ChannelDao, pero existe una importante diferencia, cuando, por ejemplo, queremos “traernos” al contacto con el id 1, es decir:
ContactDao c=new ContactDao(1);
¿Debemos traernos también todos sus contactos? Dado que existe un List en Channel que almacena su lista de Channel, parece claro que sí. Para realizar esta operación tenemos dos posibilidades:

```
SELECT * from Contact as c LEFT JOIN Channel as ch ON c.id=ch.id_contact WHERE c. id=1
```

O bien:

```
SELECT * FROM Contact WHERE id=1
SLEECT * FROM Channel WHERE id_contact=1
```

La segunda opción require dos consultas, pero su tratamiento puede ser más fácil, puesto que una consulta está destinada a Contact, porque crea un contacto, Y la segunda a Channel, por lo que crea una lista de Channel. Es importante, no solo la eficiente, sino tener un código claro, sobre todo ahora que estamos aprendiendo.

La segunda pregunta es ¿quién hace cada consulta?
Debe ser Contacto quien realice las dos peticiones. Podría ser, pero considero importante que cada DAO trate sus propios datos, por lo que la primera debe estar en Contact y la segunda en Channel (de hecho se llama findByContact). Evidentemente, la segunda será ejecutada por orden de Contact, pero el código debe estar en Channel. Una vez más, todo ordenado y separado.

La tercera pregunta es ¿cuándo hacemos las consultas?
La primera consulta está claro, en cuanto queremos instanciar al contacto, pero la segunda puede tener dos planteamientos:

- EAGER: justo después de la primera, es decir cuando instanciamos un contacto nos traemos, en ese momento todos sus canales.
- LAZY: cuando se haga un getChannel() se comprueba si están, sino están entonces se ejecuta, no antes, mientras su valor es null.

Cada aproximación tienes sus ventajas e inconvenientes. El programador decide cuál implementar. De hecho, cuando veamos JPA (el framework que realizar ORM por nostros) también indicaremos que aproximación queremos que haga. Normalmente las relaciones 1:1 se hacen con EAGER y las 1:N o N:M con LAZY para evitar sobre carga de datos que quizá no sean necesarios. Pensemos en el siguiente ejemplo, un contacto que tiene 1 millón de canales. O mejor, 1000 contactos con 1 millón de canales (e imágenes y muchos más datos cada uno) Si solo queremos listar los contactos y nada más, ¿tiene sentido hacer una consulta que "traiga" tanta información?
Dado que en nuestro caso tenemos una relación 1:N y su complejidad es mayor, hemos implementado el mecanismo LAZY.
Por eso en el constructor solo nos traemos los datos del contacto y hemos sobreescrito getChannel:

```
@Override
  public List<Channel> getChannels() {
    if (this.channels == null) {
      //lazy retrieve
      List<Channel> result = ChannelDao.getByContact(con, id);

      this.setChannels(result);
    }
    return super.getChannels();
  }
```

En caso de que aún esté a null, solicitamos la consulta y creamos la lista, aunque sea una lista vacía, pero dejará de ser null, indicando que ya se cargaron.

Otro aspecto relevante es save, no solo se debe guardar el contacto, sino asegurarse de que los canales que tiene ahora mismo esa instancia coinciden con los canales de la tabla:

```
con.setAutoCommit(false);

int rs = ConnectionUtil.execUpdate(con, q.getQ(), params, (q == queries.INSERT ? true : false));
if (q == ContactDao.queries.INSERT) {
   this.id = rs;
}
if (channels != null) {  //si se ha modificado algo sobre channels
   //Borrando aquellos que no están ya -> coherencia
   List<Channel> oldChannels = ChannelDao.getByContact(con, id);
   for (Channel oldChannel : oldChannels) {
      if (!channels.contains(oldChannel)) {
         ChannelDao cd = new ChannelDao(oldChannel);
         cd.remove();
      }
   }
   //Actualizando o insertando los nuevos
   for (Channel newChannels : channels) {
      ChannelDao cd = new ChannelDao(newChannels);
      cd.setId_contact(id);  //me aseguro de la relación
      cd.save();
   }
}

//Fin de la transacción
con.commit();
con.setAutoCommit(true);
```

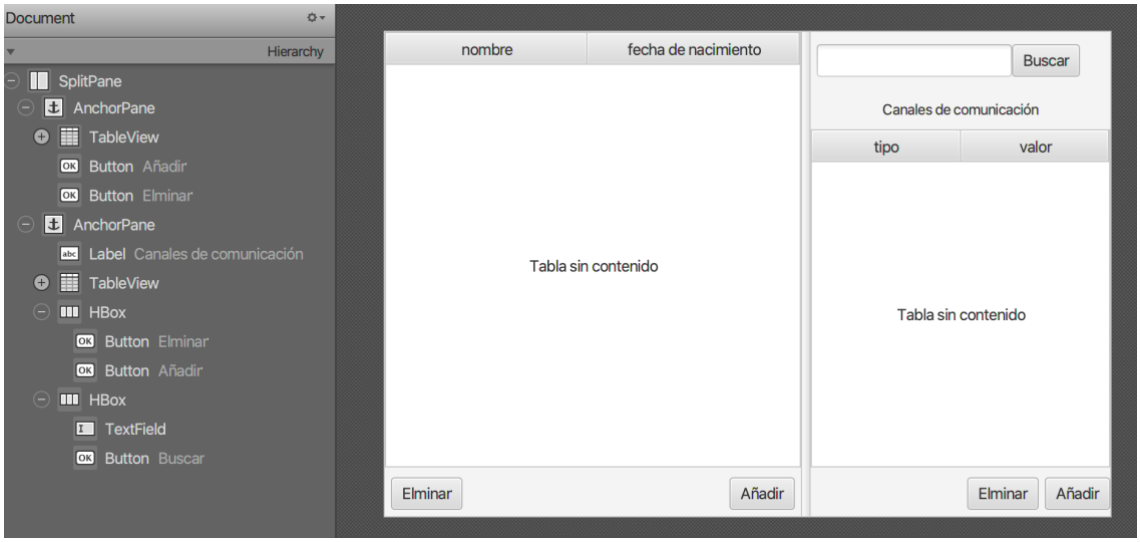
Lo hacemos todo como una transación.

1. Actualizamos o insertamos el elemento. En caso de ser inserción leemos el id recién insertado y nos lo autoasignamos.
2. En caso de que lista de canales sea distinto de null, es decir los hayamos leídos por lo que tenemos una lista actualizada de sus canales en la instancia:
    a) Borramos los canales que están en la base de datos que ya no aparecen en nuestra lista (porque lo habremos borrado).
    b) Insertamos o actualizamos los canales.

Finalmente terminamos la transacción.

Lo mismo realizamos en remove para mantener la coherencia con su tabla relacionada.

Ahora que ya tenemos la lógica de negocio preparada, toca la interfaz. Para ello comenzamos primero viendo el archivo *primary.fxml*.




Tendremos una tabla para los contactos. Dicha tabla será editable, es decir se podrá modificar directamente. En caso de hacer clic en un contacto ser verán sus canales en la tabla de la derecha. Además, contamos con un buscador que lo realizará por nombre y valor del canal.

Su código es:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.Button?>
<?import javafx.geometry.Insets?>

<SplitPane dividerPositions="0.580267558528428" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.carlosserrano.proyectojavafx.controller.PrimaryController">
 <items>
  <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0">
     <children>
       <TableView fx:id="contactTable" editable="true" layoutX="27.0" layoutY="38.0" prefHeight="398.0" prefWidth="287.0" AnchorPane.bottomAnchor="40.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
         <columns>
          <TableColumn fx:id="nameCol" minWidth="0.0" text="nombre" />
           <TableColumn fx:id="birthCol" prefWidth="231.0" text="fecha de nacimiento" />
         </columns>
         <columnResizePolicy>
           <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
         </columnResizePolicy>
       </TableView>
```

```
        <Button mnemonicParsing="false" onAction="#addContact" text="Añadir" AnchorPane.bottomAnchor="5.0" AnchorPane.rightAnchor="5.0" />
        <Button fx:id="deleteContact" layoutY="365.0" mnemonicParsing="false" onAction="#removeContact" text="Elminar" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="5.0" />
      </children></AnchorPane>
    <AnchorPane prefHeight="398.0" prefWidth="144.0">
      <children>
        <Label layoutX="59.0" layoutY="56.0" text="Canales de comunicación" AnchorPane.topAnchor="56.0">
          <font>
            <Font size="12.0" />
          </font>
        </Label>
        <TableView fx:id="channelTable" editable="true" layoutX="-7.0" layoutY="62.0" prefHeight="398.0" prefWidth="162.0" AnchorPane.bottomAnchor="40.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="80.0">
          <columns>
            <TableColumn fx:id="typeCol" prefWidth="72.0" text="tipo" />
            <TableColumn fx:id="valueCol" prefWidth="78.0" text="valor" />
          </columns>
          <columnResizePolicy>
            <TableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
          </columnResizePolicy>
        </TableView>
        <HBox fx:id="menuChannel" layoutX="3.0" layoutY="366.0" AnchorPane.rightAnchor="0.0">
          <children>
            <Button fx:id="deleteChannel" layoutX="48.0" layoutY="371.0" mnemonicParsing="false" onAction="#removeChannel" text="Elminar" AnchorPane.bottomAnchor="5.0" AnchorPane.leftAnchor="3.0">
              <HBox.margin>
                <Insets right="5.0" />
              </HBox.margin>
            </Button>
            <Button layoutX="48.0" layoutY="371.0" mnemonicParsing="false" onAction="#addChannel" text="Añadir" AnchorPane.bottomAnchor="5.0" AnchorPane.rightAnchor="3.0" />
          </children>
        </HBox>
        <HBox layoutX="14.0" layoutY="22.0" AnchorPane.leftAnchor="5.0" AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="10.0">
          <children>
            <TextField fx:id="searchPattern" layoutX="14.0" layoutY="22.0" />
            <Button layoutX="175.0" layoutY="22.0" mnemonicParsing="false" onAction="#search" text="Buscar" />
          </children>
        </HBox>
      </children>
    </AnchorPane>
  </items>
</SplitPane>
```

Su controlador es *PrimaryController.java* que ha sufrido numerosos cambios.
Es un controlador muy importante, puesto que no solo trata con los DAO, sino que permite a la interfaz tener tablas editables, haciendo su uso muy sencillo:

```
package com.carlosserrano.proyectojavafx.controller;
```

```
import com.carlosserrano.proyectojavafx.model.Channel;
import com.carlosserrano.proyectojavafx.model.Contact;
import com.carlosserrano.proyectojavafx.model.dao.ChannelDao;
import com.carlosserrano.proyectojavafx.model.dao.ContactDao;
import com.carlosserrano.proyectojavafx.utils.ConnectionUtil;
import java.net.URL;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.ResourceBundle;
import java.util.Set;
import javafx.beans.property.SimpleObjectProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.layout.HBox;

public class PrimaryController extends Controllers implements Initializable {

    //OBSERVABLE <--------
    public ObservableList<Contact> contacts;
    public ObservableList<Channel> channels;

    @FXML
    private TableView<Contact> contactTable;

    @FXML
    private TableColumn<Contact, String> nameCol;
    @FXML
    private TableColumn<Contact, String> birthCol;

    @FXML
    private TableView<Channel> channelTable;

    @FXML
    private TableColumn<Channel, String> typeCol;
    @FXML
    private TableColumn<Channel, String> valueCol;

    private java.sql.Connection con;

    @FXML
    private TextField searchPattern;
    @FXML
    private Button deleteContact;
    @FXML
    private Button deleteChannel;
```

```
@FXML
private HBox menuChannel;


@Override
public void initialize(URL url, ResourceBundle rb) {
   this.contacts = FXCollections.observableArrayList();

   //loading
   this.con = ConnectionUtil.connect(AppController.currentConnection);
   if (con != null) {
      List<Contact> c = ContactDao.getAll(con);
      this.contacts.addAll(c);
   }

   nameCol.setCellValueFactory(cellData -> {
      return new SimpleObjectProperty<>(cellData.getValue().getNickname());
   });
   nameCol.setCellFactory(TextFieldTableCell.forTableColumn());
   nameCol.setOnEditCommit(
         new EventHandler<TableColumn.CellEditEvent<Contact, String>>() {
      @Override
      public void handle(TableColumn.CellEditEvent<Contact, String> t) {

         Contact selected = (Contact) t.getTableView().getItems().get(
               t.getTablePosition().getRow());
         selected.setNickname(t.getNewValue());
         ContactDao cc = new ContactDao(selected);
         cc.save();
      }
   }
   );

   birthCol.setCellValueFactory(cellData -> {
      return new SimpleObjectProperty<>(cellData.getValue().getBirthDate().toString());
   });
   birthCol.setCellFactory(TextFieldTableCell.forTableColumn());
   birthCol.setOnEditCommit((TableColumn.CellEditEvent<Contact, String> t) -> {
      Contact selected = (Contact) t.getTableView().getItems().get(
            t.getTablePosition().getRow());
      selected.setBirthDate(LocalDate.parse(t.getNewValue()));  //habría que validar
      ContactDao cc = new ContactDao(selected);
      cc.save();
   });

   contactTable.setEditable(true);

   contactTable.getSelectionModel().cellSelectionEnabledProperty().set(true);

   contactTable.getSelectionModel().selectedItemProperty().addListener(
         (observable, oldValue, newValue) -> showChannels(newValue)
   );
   //Add observable
   contactTable.setItems(contacts);

   this.channels = FXCollections.observableArrayList();
```

```
    typeCol.setCellValueFactory(cellData -> {
        return new SimpleObjectProperty<>(cellData.getValue().getType());
    });
    typeCol.setCellFactory(TextFieldTableCell.forTableColumn());
    typeCol.setOnEditCommit((TableColumn.CellEditEvent<Channel, String> t) -> {
        Channel selected = (Channel) t.getTableView().getItems().get(
                t.getTablePosition().getRow());
        selected.setType(t.getNewValue());
        ChannelDao cc = new ChannelDao(selected);
        cc.save();
    });
    valueCol.setCellValueFactory(cellData -> {
        return new SimpleObjectProperty<>(cellData.getValue().getValue());
    });
    valueCol.setCellFactory(TextFieldTableCell.forTableColumn());
    valueCol.setOnEditCommit((TableColumn.CellEditEvent<Channel, String> t) -> {
        Channel selected = (Channel) t.getTableView().getItems().get(
                t.getTablePosition().getRow());
        selected.setValue(t.getNewValue());  //habría que validar
        ChannelDao cc = new ChannelDao(selected);
        cc.save();
    });

    channelTable.setEditable(true);

    channelTable.getSelectionModel().cellSelectionEnabledProperty().set(true);

    channelTable.setItems(channels);
    showChannels(null);

    searchPattern.setOnAction(new EventHandler<ActionEvent>(){
        @Override
        public void handle(ActionEvent t) {
            search();  //<--searchPattern.getText()
        }

    });

}

private void showChannels(Contact c) {
    channels.clear();
    if (c != null) {
        List<Channel> lc = ChannelDao.getByContact(con, c.getId());
        channels.addAll(lc);
        menuChannel.setDisable(false);
        deleteContact.setDisable(false);
    }else{
        menuChannel.setDisable(true);
        deleteContact.setDisable(true);
    }
}

@Override
void onLoad() {
    this.app.controller.title("CRUD - JAVAFX");
    this.app.controller.enableCon();
```

```
    }

    @FXML
    public void addContact() {
        Contact nuevo=new Contact();
        ContactDao nuevoDao=new ContactDao(nuevo);
        nuevoDao.save();
        nuevo.setId(nuevoDao.getId());
        contacts.add(nuevo);
    }

    @FXML
    public void removeContact() {
        Contact selected = contactTable.getSelectionModel().getSelectedItem();
        if (selected != null) {
            contacts.remove(selected);
            ContactDao cc = new ContactDao(selected);
            cc.remove();
        }
    }

    @FXML
    public void addChannel() {
        Contact selected = contactTable.getSelectionModel().getSelectedItem();
        if (selected != null) {
            Channel nc = new Channel();
            nc.setType("email");
            nc.setId_contact(selected.getId());
            ChannelDao ncdao=new ChannelDao(nc);
            ncdao.save();
            nc.setId(ncdao.getId());
            channels.add(nc);
        }
    }

    @FXML
    public void removeChannel() {
        Contact selected = contactTable.getSelectionModel().getSelectedItem();
        if (selected != null) {
            Channel selectedC = channelTable.getSelectionModel().getSelectedItem();
            if (selectedC != null) {
                channels.remove(selectedC);
                (new ChannelDao(selectedC)).remove();
            }
        }
    }

    @FXML
    public void search(){
        String pattern=searchPattern.getText();
        pattern=pattern.trim();
        contacts.clear();
        List<Contact> newC;
        if(pattern.equals("")){
            newC=ContactDao.getAll(con);
        }else{
            newC=ContactDao.getByName(con, pattern);
```

```
        }

        //Extra funcion -> buscar también por contacto

        List<Contact> newC2=new ArrayList<>();
        if(!pattern.equals("")){
            List<Channel> lc=ChannelDao.getByValue(con, pattern);
            Set<Integer> listId=new HashSet<>();
            lc.stream().forEach(c->{
                listId.add(c.getId_contact());
            });
            if(listId.size()>0){
                newC2=ContactDao.getById(con, new ArrayList(listId));
            }
        }
        contacts.addAll(newC);
        for(Contact c:newC2){
            if(!contacts.contains(c)){
                contacts.add(c);
            }
        }
    }

}
```

Tanto la tabla de contactos como la tabla de canales estarán asociadas a listas observables como vimos en la pantalla de conexiones. Analicemos como se ha podido hacer una tabla editable:

```
contactTable.setEditable(true);
contactTable.getSelectionModel().cellSelectionEnabledProperty().set(true);
```

Estas dos líneas permiten que la tabla sea editable y que las celdas sean seleccionables de forma individual, no por fila completa.

De igual forma hacemos que se rellenen con el campo adecuado. En este caso, dado que los objetos tienen campos normales (String), debemos convertirlos en observables (SimpleObjectProperty) para que si cambian en el controlador la vista se refresque.

```
birthCol.setCellValueFactory(cellData -> {
            return new SimpleObjectProperty<>(cellData.getValue().getBirthDate().toString());
        });
```

Asignamos un Textfield cuando se active la edición para poder modificar el valor. Es interesante que existen otras posibilidades como son combobox, textarea…

```
        birthCol.setCellFactory(TextFieldTableCell.forTableColumn());
```

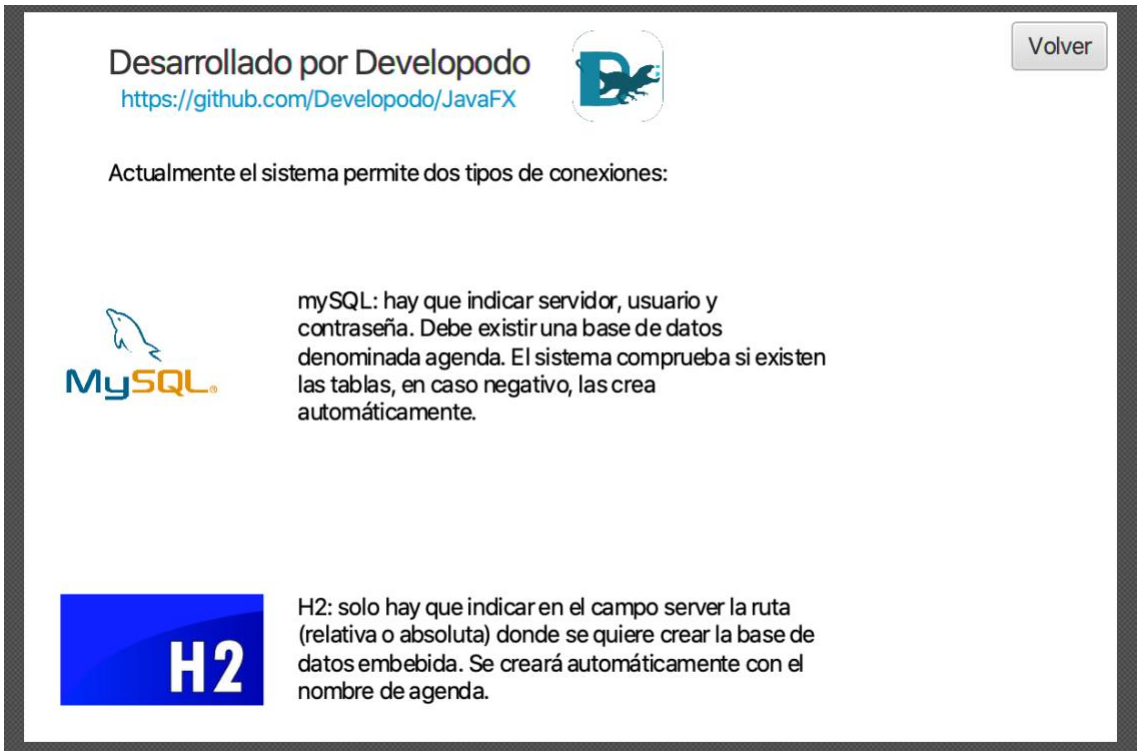
Cuando se finalice la edición, asociamos un evento para realizar la actualización en la base de datos mediante el DAO. Es decir, un cambio en la interfaz, genera un cambio en el controlador que invoca un cambio en DAO o lo que es lo mismo en la base de datos.

```
        birthCol.setOnEditCommit((TableColumn.CellEditEvent<Contact, String> t) -> {
```

```
        Contact selected = (Contact) t.getTableView().getItems().get(
              t.getTablePosition().getRow());
        selected.setBirthDate(LocalDate.parse(t.getNewValue()));  //habría que validar
        ContactDao cc = new ContactDao(selected);
        cc.save();
    });
```

Es interesante el buscador que permite busca contactos por nombres y por el valor de sus canales.

Hemos modificado *about.fxml*:

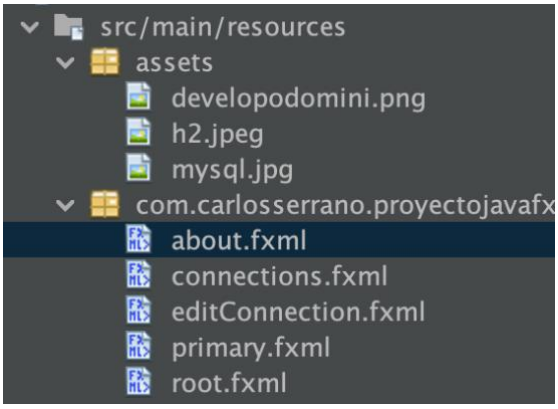



Su código es:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.image.*?>
<?import javafx.scene.web.*?>
<?import javafx.scene.text.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane id="AnchorPane" prefHeight="400.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="com.carlosserrano.proyectojavafx.controller.AboutController">
  <children>
    <Button layoutX="545.0" layoutY="1.0" mnemonicParsing="false" onAction="#back" text="Volver"
AnchorPane.rightAnchor="5.0" AnchorPane.topAnchor="5.0" />
```

```
    <Label layoutX="46.0" layoutY="15.0" text="Desarrollado por Developodo" AnchorPane.leftAnchor="46.0" AnchorPane.topAnchor="15.0">
      <font>
        <Font size="19.0" />
      </font>
    </Label>
    <Text layoutX="154.0" layoutY="138.0" strokeType="OUTSIDE" strokeWidth="0.0" text="mySQL: hay que indicar servidor, usuario y contraseña. Debe existir una base de datos denominada agenda. El sistema comprueba si existen las tablas, en caso negativo, las crea automáticamente." wrappingWidth="294.03369140625" AnchorPane.leftAnchor="150.0" AnchorPane.topAnchor="150.0" />
    <Text layoutX="46.0" layoutY="92.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Actualmente el sistema permite dos tipos de conexiones:" AnchorPane.topAnchor="80.0" />
    <Text layoutX="90.0" layoutY="255.0" strokeType="OUTSIDE" strokeWidth="0.0" text="H2: solo hay que indicar en el campo server la ruta (relativa o absoluta) donde se quiere crear la base de datos embebida. Se creará automáticamente con el nombre de agenda." wrappingWidth="294.03369140625" AnchorPane.bottomAnchor="20.0" AnchorPane.leftAnchor="150.0" />
    <ImageView fitHeight="68.0" fitWidth="89.0" layoutX="46.0" layoutY="133.0" pickOnBounds="true" preserveRatio="true" AnchorPane.leftAnchor="20.0" AnchorPane.topAnchor="160.0">
      <image>
        <Image url="@../../../../assets/mysql.jpg" />
      </image>
    </ImageView>
    <ImageView fitHeight="61.0" fitWidth="122.0" layoutX="457.0" layoutY="240.0" pickOnBounds="true" preserveRatio="true" AnchorPane.bottomAnchor="20.0" AnchorPane.leftAnchor="20.0">
      <image>
        <Image url="@../../../../assets/h2.jpeg" />
      </image>
    </ImageView>
    <ImageView fitHeight="50.0" fitWidth="59.0" layoutX="301.0" layoutY="14.0" pickOnBounds="true" preserveRatio="true" AnchorPane.topAnchor="10.0">
      <image>
        <Image url="@../../../../assets/developodomini.png" />
      </image>
    </ImageView>
    <Hyperlink fx:id="repo" layoutX="48.0" layoutY="35.0" text="https://github.com/Developodo/JavaFX" />
  </children>
</AnchorPane>
```

Hemos creado una carpeta assets en resources con tres imágenes que empleamos en esta vista:



# 7 GENERAR JAR

Existen multiples formas de paquetizar nuestro Proyecto. En este caso haremos un jar con todas las dependecias embebidas (librerías, assets, fxml… todo lo necesario para ejecutar nuestra app). Para ello hay que realizar algunos cambios.

En el POM añadiremos un plugin para generar este jar, quedando *pom.xml* de la siguiente forma:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.carlosserrano</groupId>
  <artifactId>ProyectoJavaFX</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>Demo - JavaFX - JAXB - MYSQL - H2</name>
  <url>https://github.com/Developodo/JavaFX</url>
  <developers>
    <developer>
      <id>Developodo</id>
      <name>Developodo</name>
      <email>developodo@gmail.com</email>
    </developer>
  </developers>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.build.finalName>JavaFX-Demo</project.build.finalName>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <javafx.version>14</javafx.version>
    <mainClass>com.carlosserrano.proyectojavafx.Executable</mainClass>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-controls</artifactId>
      <version>14</version>
    </dependency>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-fxml</artifactId>
      <version>14</version>
    </dependency>
    <dependency>
      <groupId>jakarta.xml.bind</groupId>
      <artifactId>jakarta.xml.bind-api</artifactId>
      <version>2.3.3</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.glassfish.jaxb/jaxb-runtime -->
    <dependency>
      <groupId>org.glassfish.jaxb</groupId>
```

```
        <artifactId>jaxb-runtime</artifactId>
        <version>2.3.2</version>
    </dependency>
        <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.19</version>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <release>11</release>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-maven-plugin</artifactId>
        <version>0.0.4</version>
        <configuration>
          <mainClass>com.carlosserrano.proyectojavafx.App</mainClass>
        </configuration>
      </plugin>

      <!-- to BUILD -->
      <plugin>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.1</version>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                  <mainClass>${mainClass}</mainClass>
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
    <finalName>DemoJavaFX</finalName>
  </build>
```

```
</project>
```

Hemos añadido información del desarrollador, creado variables para controlar la versión de javafx o la clase principal. Y sobre todo incluido el plugin maven-shade que genera DemoFavaFX.jar.

Solo falta un pequeño detalle. Debemos crear el punto de entrada en una clase que no extienda de Application, por ello en el paquete *com.carlosserrano.proyectojavafx* creamos una clase *Executable.java*:

```
package com.carlosserrano.proyectojavafx;

public class Executable {
   public static void main(String[] args) {
      App.main(args);
   }

}
```

Básicamente ejecuta el main original de nuestra clase App. Esta clase Executable debe ser la establecida en el pom.xml como punto de entrada.
Ahora contruimos nuestra app:

Nos vamos a la ruta donde se ha generado nuestro jar, normalmente en {ruta de nuestro proyecto}/target/DemoJavaFX.jar y lo copiamos a donde deseemos almacenar nuestra app ejecutable.
Y finalmente en un terminal ejecutamos (si nos encontramos en la misma carpeta que el jar):

```
java -jar DemoJavaFX.jar
```

Voilà!! FIN!
El zip completo del proyecto entero se encuentra en:
https://github.com/Developodo/JavaFX/raw/master/com.carlosserrano_ProyectoJavaFX_FINAL.zip
El jar compilado:
https://github.com/Developodo/JavaFX/raw/master/DemoJavaFX.jar

# 8 EJERCICIO PROPUESTO

1. Insertar un loading mientras se hacen las consultas dado que la interfaz puede quedar bloqueada hasta que acaba…
2. Crear una relación 1 a 1 con Contact para que almacene una foto suya en otra tabla. Buscar info de cómo se puede almacenar un archivo en SQL. Buscar cómo se cogen archivos desde el disco duro (FileChooser) y estudiar la posibilidad de acceder a una cámara.
3. Validar la fecha de nacimiento, insertar un DatePicker para insertarla.
4. Permitir exportar una base de datos -> serializar la lista a archivo. Permitir importar de un archivo (serializado) una serie de contactos a la base de datos.

5. Convertir el paquete en un archivo ejecutable y con icono personalizado.